(1)

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GE/ENG/88D-23

PROBLEM SPECIFIC APPLICATIONS FOR NEURAL NETWORKS

THESIS

Mark K. Lutey
Captain, USAF

AFIT/GE/ENG/88D-23

Approved for public release; distribution unlimited

AFIT/GE/ENG/88D-23

PROBLEM SPECIFIC APPLICATIONS FOR NEURAL NETWORKS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Mark K. Lutey
Captain, USAF

December 1988

Acknowledgements

This work is dedicated to my loving wife Joan and my son David who spent the many hours alone while this thesis work was conducted. Also, I wish to thank Dr. Matthew Kabrisky and Capt Steve Rogers for their encouragement and inspiration which made this thesis possible. I shall remember them always.

Table of Contents

List Of Figures

List of Tables

Abstract

The purpose of this thesis is to examine several topics relating to neural networks. First, the investigation of the error output for a multi-layer perceptron is examined to determine if the error calculation can be modified to decrease the training time. The result is a slight improvement.

Next, the sigmoid function usually used by multi-layer perceptrons is investigated to determine if modifying terms within the sigmoid function will decrease training time. An improvement is found in the performance and in some cases by much as an order of magnitude.

Then the subject of adding an additional class to the problem space is examined. Regardless of the data class added or the network size, there is no advantage to using a previously trained multi-layer perceptron as a starting state to be trained additionally to include the new data class. Nothing is gained compared to starting the network form its untrained state.

This is followed by an investigation to determine whether a multi-layer perceptron can be used to reduce noise added to a signal. Results show that the multi-layer perceptron, when trained with three specific frequencies, reduced noise but would "resonate" at these frequencies

only. The network may also be limited to the number of individual frequencies on which it will perform noise reduction.

Next, the combination of using a Kohonen self-organizing feature map and a multi-layer perceptron to perform isolated word recognition is tested. An 80 per cent accuracy is achieved for speaker dependent, isolated word recognition. Accuracy falls to approximately 40 per cent for speaker independent, isolated word recognition.

Finally, an improvement in the basic neuron element usually used by the single and multi-layer perceptrons is presented. Results show that the computational power for a single neuron is greatly enhanced and that use in a multi-layer perceptron configuration is still possible.

PROBLEM SPECIFIC APPLICATIONS FOR NEURAL NETWORKS

I. Introduction

Man is capable of performing feats of calculation, such as speech recognition, that easily surpass the performance of today's super-computers. How the human brain accomplishes this feat is still not understood. But, over the last 40 years, researchers have made some progress in understanding how the human brain may operate. Some of the researchers are looking at the neurons (that is, the nerve cells) of which the human brain is mainly composed. One result of their studies is the analytical modelling of how neurons, or a collection of neurons, may operate. These models have led to the development of the area of research called "neural networks".

Since neural networks mimic the parallel connections found in the human brain, they provide the capability to do calculations in parallel that can be simulated by today's typical serial-type computer. Also, like the human brain, damage to a small number of neurons will not disable the whole network, unlike damage to a computer which could mean complete failure. Thus, these neural networks seem to push us closer to achieving the capabilities that are provided by

the human brain.

Unfortunately, since this area of research is relatively new, the extent of the models' capability is not fully understood. Therefore, more research in the area of neural networks will be necessary before these models have the possibility of achieving the computational feats of man.

Background

Neural networks currently use a variety of ways to model neurons by employing either electrical circuits or mathematical models used by computers, as shown in Figure 1. To

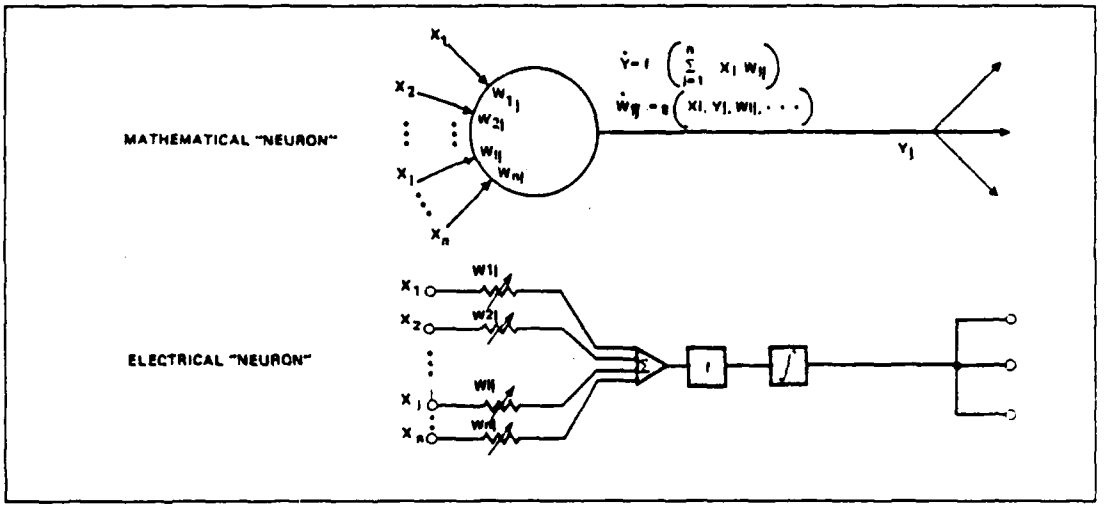


Figure 1. Neural Network Models. (From 10:211).

mimic neurons, the networks control how much the neuron responds to an input, to how many other neurons any one neuron will be connected, and what the strength of its synaptic connections (the area of contact between neurons) will be.

Typically, it is known how to control the responses and

how the neurons will be connected; what is not always known are the strengths of the synaptic connections between the neurons. Depending on the problem that is to be solved, these strengths (or weights) will need to be adjusted to find the solution to a particular problem. Thus, these neurons are put through a training period that will allow the neurons to change their weights as required by the problem. Some networks do not require a training period since they do not change any of their initial conditions and these are calculated before the networks are used in problem solving (5:7). Training is accomplished by presenting a set of chosen inputs (data) repeatedly until the network changes its weights to achieve the desired output.

Training of the network can be supervised or unsupervised (5:6). If the networks are trained under supervision, they must be told how to change their weights by the user. If the network is to be unsupervised in training, then the network makes all the decisions on how to change the weights among the neurons. Also, some networks allow for continuous training while the network is in use.

Three of the most well-known neural networks are the single-layer perceptron, the multi-layer perceptron, and the Kohonen self-organizing feature map (5:6). The basic differences between these neural networks are whether they need supervised or unsupervised training. All three of these networks can accept discrete (binary) or continuous inputs (5:6).

Single-Layer Perceptron. The single-layer perceptron (shown in Figure 2) uses either discrete or continuous input data and requires supervised training (5:13). The single neuron can make decisions only between two separate data classes. It accomplishes this by creating a hyperplane to separate the two data classes and therefore acts as a decision boundary, shown in Figure 3. If the two classes begin to overlap, the single-layer perceptron will have trouble deciding to which class an input belongs.

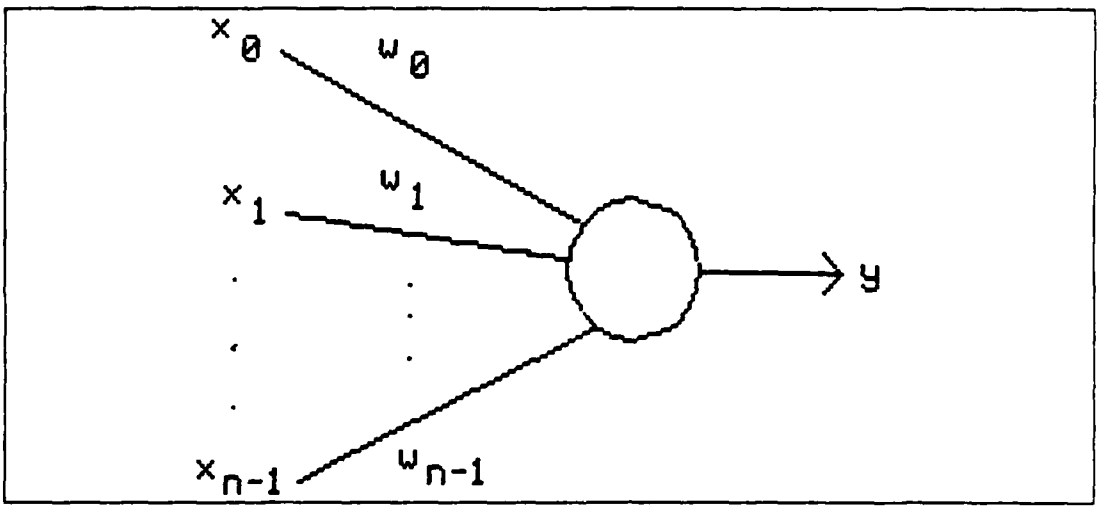


Figure 2. A Single-Layer Perceptron (From 5:13).

Multi-Layer Perceptron. The multi-layer perceptron (shown in Figure 4) uses either continuous or discrete inputs and requires supervised training (5:15). This network is composed of single-layer perceptrons layered in rows between the input and the output neurons. Since the middle layers cannot see the outputs, they are called the hidden layers (5:17). Errors are calculated at the outputs and are propagated down to each of the neurons in the layers below.

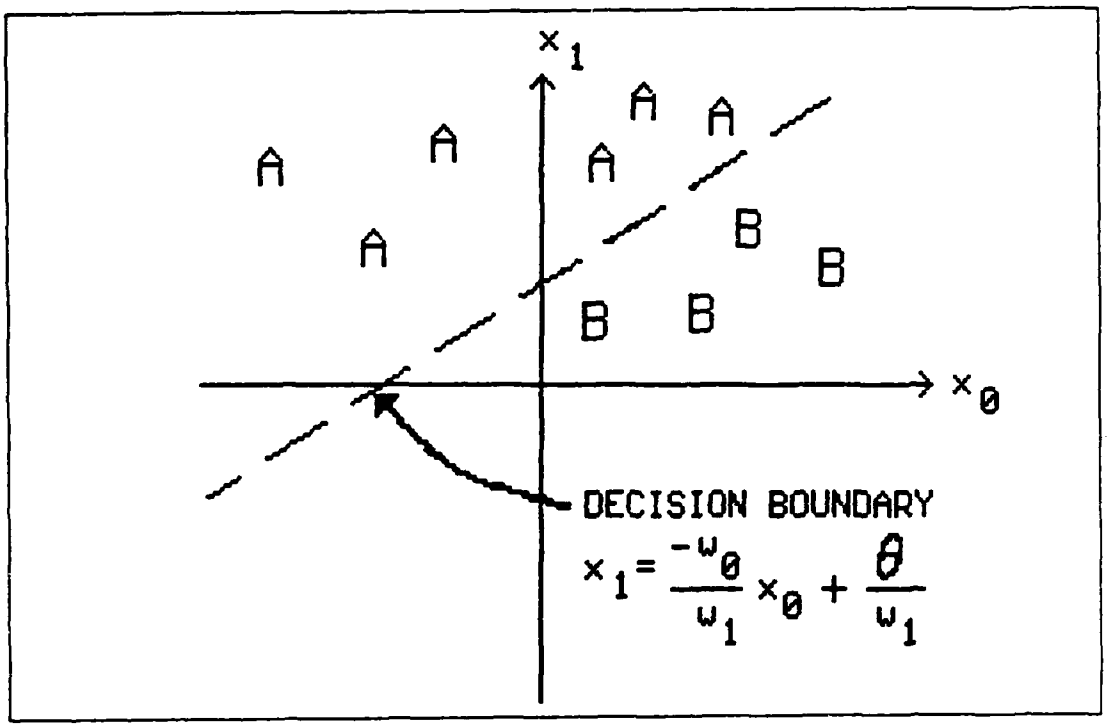


Figure 3. Decision Boundary Placement (From 5:13).

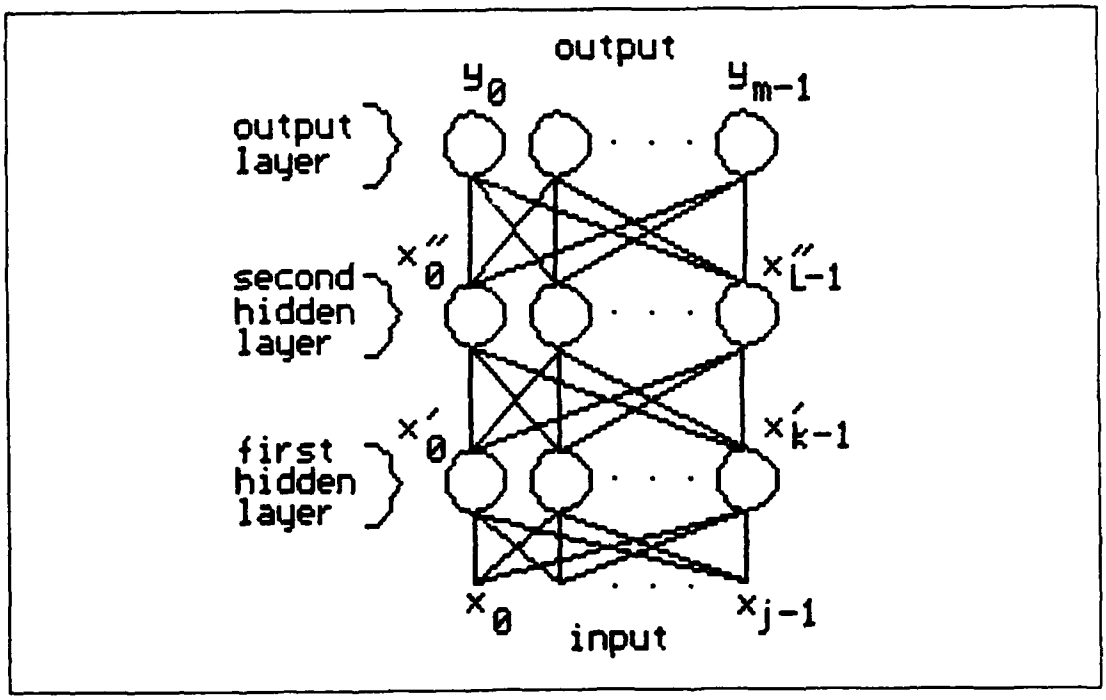


Figure 4. A Multi-Layer Perceptron (From 5:16).

This downward propagation of the errors allows the neurons in the hidden layers to adjust the weights.

This network, unlike the single-layer perceptron, can distinguish more than two classes (5:16). It accomplishes this by creating numerous hyperplanes that can make boundaries surrounding each data class if necessary. As the boundaries are made, the network determines whether the data classes are disjoint (broken into pieces). If so, all the separate regions belonging to the same class are joined together.

However, one problem with the multi-layer perceptron is that it usually requires more extensive training than that experienced by a single-layer perceptron (5:18).

Kohonen Self-Organizing Feature Map. The Kohonen self-organizing feature map (or Kohonen net) can use either continuous or discrete inputs and allows unsupervised training. This network differs from the single and multi-layer perceptrons, in that every neuron in the network can see the inputs, as shown in Figure 5. As the training progresses, the neurons will "decide" among themselves which neuron or group of neurons (called a neighborhood) best represents the input (4:18). Thus, if the class to which an input belongs is not known beforehand, this network will group that input with other similar inputs to a certain area within the network. This network is called self-organizing because of this intrinsic proclivity to group of similar inputs (4:13).

It has been suggested that the Kohonen net may not only

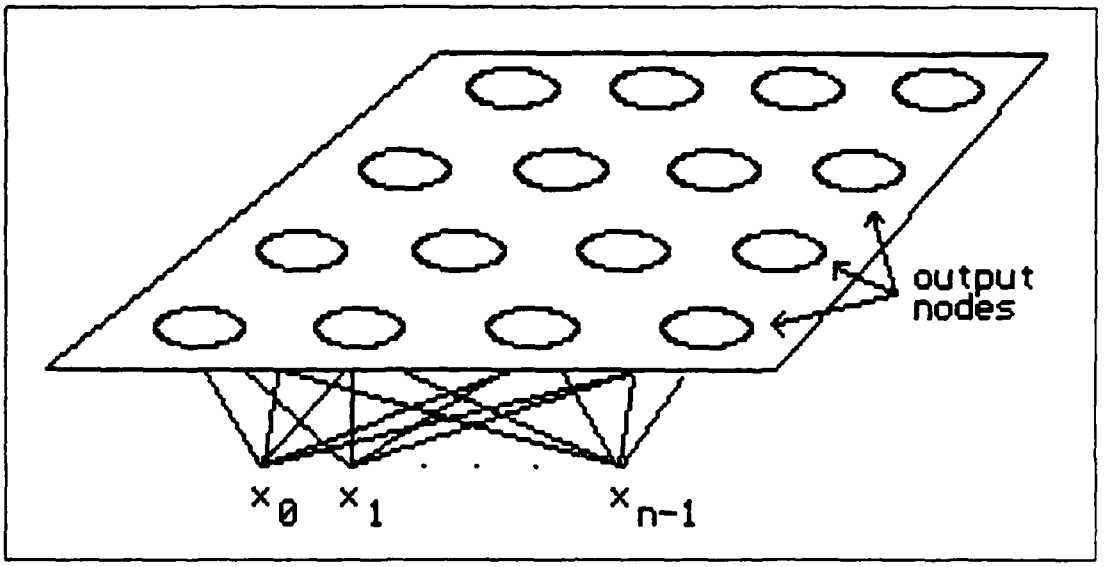


Figure 5. A Kohonen Self-Organizing Feature Map. (From 5:18)

model how neurons work, but may also model how the brain tends to map certain inputs to special areas of the brain (4:10). This network also performs extremely well with noisy inputs, since any input will tend to excite at least one group of neurons (4:19).

A word of caution is in order, however, since virtually no brain mechanisms are understood outside of the sensory/motor mapping regions and a person would have to be foolhardy or ignorant to claim that one neural network or another actually behaves like real brain circuits.

Definitions

Discriminative Functions. This term is used collectively to explain how a neuron will respond in terms of the sum of its weighted inputs. The discriminative function may be any one of three nonlinear, positively increasing func-

tions that are used in neural networks, as shown in Figure 6. Note that the outputs of these functions may vary; for instance, the hard limiter function can range from zero to one if desired.

Threshold. An input, in addition to the weighted inputs, used to help form the decision boundary between two classes. The threshold is tied to a constant value and may or may not undergo training.

Problem

The primary purpose of this thesis is to examine several questions:

1) Is there a method to reduce the training time for a multi-layer perceptron by adjusting the output error calculations? (Chapter II)

2) Can adjusting the sigmoid function reduce the training time for a multi-layer perceptron? (Chapter III)

3) Can more classes be added to a trained multi-layer perceptron with the desired result being the reduction of the overall training time? (Chapter IV)

4) Is a multi-layer perceptron capable of reducing noise in a signal? (Chapter V)

5) Can a Kohonen Neural Network and a multi-layer perceptron work together to perform isolated word recognition? (Chapter VI)

6) Can the basic neuron model be improved and still be

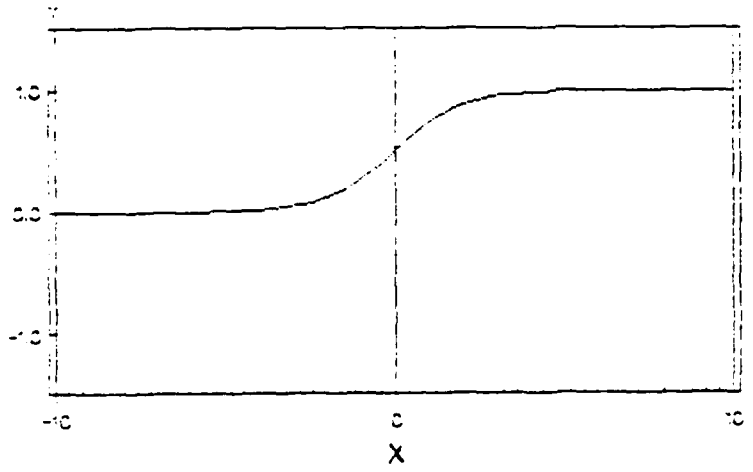


Figure 6a. Sigmoid Function.

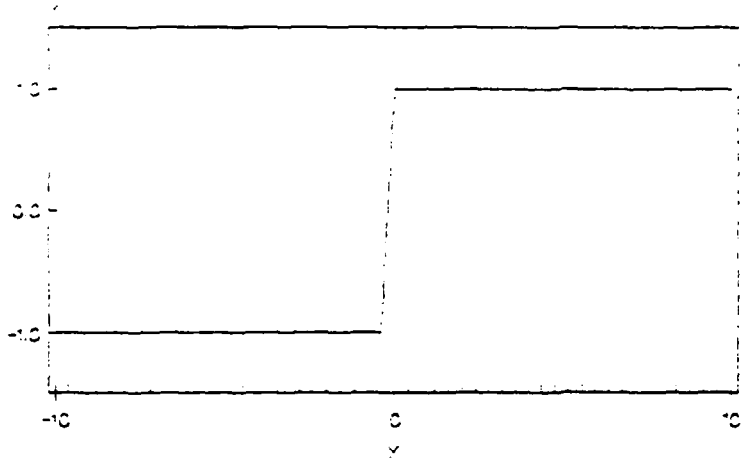


Figure 6b. Hard Limiter.

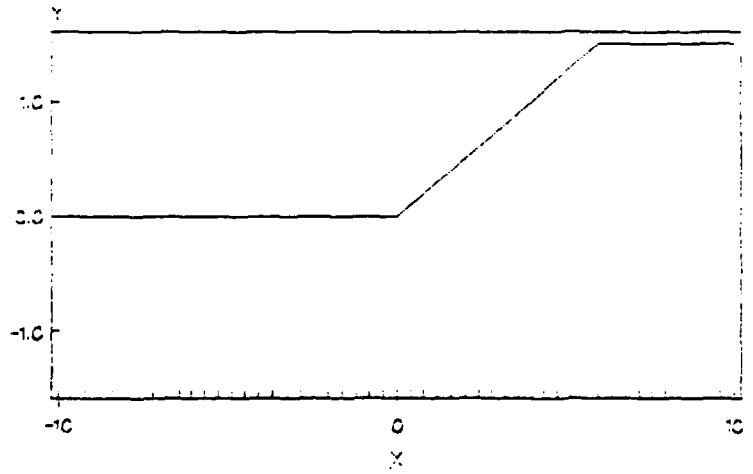


Figure 6c. Threshold Logic.

used in a multi-layer perceptron configuration? (Chapter VII)

Scope

In chapters two and three, data from a previous thesis by Captain Dennis Ruck (11) is used. It was found that there were errors when raw data was not correctly normalized before being used as inputs to a neural network. Since the intention was not optimization of performance, the data was left as originally normalized.

In chapter four, additional class training is examined using only two dimensional data in order to facilitate the visualization of the placement of hyperplanes. Also, since the multi-layer perceptron makes similar calculations for all of its neurons (and thus the hyperplane placement) regardless of the dimensionality of the problem, the results are applicable for higher dimensions. Data from the Ruck thesis is not used in this application, since the way this data is distributed in higher dimensions is not known.

For chapter five, only one network size is to be used. The intent of this chapter is to examine whether or not noise reduction in an analog, one dimensional signal can be accomplished using a multi-layer perceptron.

The intent of chapter six is to examine whether or not isolated word recognition is possible by combining the special properties found in a Kohonen self-organizing feature map and a multi-layer perceptron. No attempt is to be made

to segment words; therefore, continuous-speech recognition will not be examined.

Finally, in chapter seven, changes in the basic neuron are presented. Testing is limited to a few examples to determine if the new training rules are favorable to real applications.

Approach

To answer the first problem question, the calculation of the errors that will be propagated down through the multi-layer perceptron are examined. Next, a more desirable error calculation will be developed based on the previous analysis. Finally, the new error calculation will be tested to determine whether or not training time is improved.

For question two, the sigmoid function is examined to determine its effect on the training time. Next, the sigmoid function is adjusted and equations are developed to account for the change in the sigmoid function. Finally, the network is tested to again determine whether or not training time is improved.

To answer question three, a multi-layer perceptron is first trained for two simple data classes. More neurons are then added to the network and a new class is introduced to the training data. Then each network's performance is monitored and the training time is compared to that of a new network originally trained for all three classes. Several sizes and techniques are used to see if any reduction in

training time occurs.

In answering question four, a multi-layer perceptron is constructed based on the work of Tamura and Waibul (13). Testing of the network takes place after it is trained using a sum of three sine waves (of different frequencies) with noise added. Next, a network trained using digitized samples of speech with noise added is tested. Finally, the results are examined to determined the noise reduction capability of a multi-layer perceptron.

To answer question five, a Kohonen self-organizing feature map is trained to store spectrograms of digitized speech from an individual speaker. Afterwards, a multi-layer perceptron is trained using a "map" of the Kohonen network. The two networks are then connected and tested to determine if the combination of the two networks will perform speaker dependent and independent speech recognition.

In answering question six, the basic neuron is examined and changes are made to its structure. Next, equations are presented to allow the new neuron model to train. Finally, the new neuron model is tested to measure the computational power.

Sequence of Presentation

Chapter two investigates the output error calculations employed by a multi-layer perceptron. A new error calculation is then presented and tests are performed to determine its effectiveness in reducing training time. Finally, con-

clusions and recommendations are provided.

Chapter three examines the sigmoid function in general. Terms in the sigmoid function previously ignored are re-introduced and equations are presented for a multi-layer perceptron based on the developed terms. Tests are performed to determine if a decrease in training time occurs. Finally, conclusions and recommendations are provided.

Chapter four discusses the problem of using a partially trained multi-layer perceptron to decrease training time when a new class is introduced. Tests are performed for several multi-layer perceptron networks of various sizes. Also, different data classes are presented to determine whether or not a partially trained multi-layer perceptron can decrease training time. Finally, conclusions and recommendations are presented.

Chapter five describes the basic idea of using a multi-layer perceptron to perform noise reduction in a signal. The two types of training that are to be performed are discussed. A quick description of the testing follows. Finally, results and recommendations are presented.

Chapter six discusses the system constructed of the Kohonen net and the multi-layer perceptron and how it will perform isolated word recognition. Then the training of the two networks is described. There is a brief discussion on the testing to be performed. Finally, the results and future recommendations are provided.

Chapter seven begins by describing the changes that are

made to the structure of the neuron. Then the training rules for this new neuron are presented for use in both a single-layer and multi-layer perceptron configuration. Next, the added capabilities that are provided by the new neuron are examined. Then a description of the testing that is to be performed is presented. Finally, the results and recommendations are given.

II. Accelerated Learning for the Multi-Layer Perceptron by Modifying the Output Error

Introduction

The purpose of this chapter is to develop an acceleration rule based on the examination of the output error for a multi-layer perceptron. The first section in this chapter describes the multi-layer perceptron training rule for the output layer. The next section quickly shows the development of the acceleration rule by modifying the output error. The third and fourth sections briefly describes the testing and the results. An unexpected result found during testing is discussed in the final section.

Multi-Layer Perceptron Training Algorithm

As mentioned earlier, the multi-layer perceptron may take an extremely long time to train its weights. In the training algorithm, one requirement is that the output neurons update their weights by the equation (3:137):

$$w_{ij}(t+1) = w_{ij}(t) + \eta\ \delta_j\ x'_i \qquad (1)$$

where w_{ij} is the weight from node i in the hidden layer below to the output node j at time t, η is the gain (typically 0.2 to 0.4), δ_j is the calculated error at the output node j, and x'_i is the input to the output node j on the weight w_{ij}. The error term δ_j for the output node is given

by the equation (3:13):

$$\delta_j = y_j\ (1 - y_j)\ (d_j - y_j) \qquad (2)$$

where y_j is the actual output at node j and d_j is the desired output at node j. Also, by using a regular sigmoid function, y_j is limited to values between zero and one and therefore d_j is either zero or one.

As can be seen by equation (2), when y_j approaches zero the calculated error term goes to zero. Likewise, if the output approaches one the calculated error goes to zero. Figure 7 shows the calculated error for y_j over the range

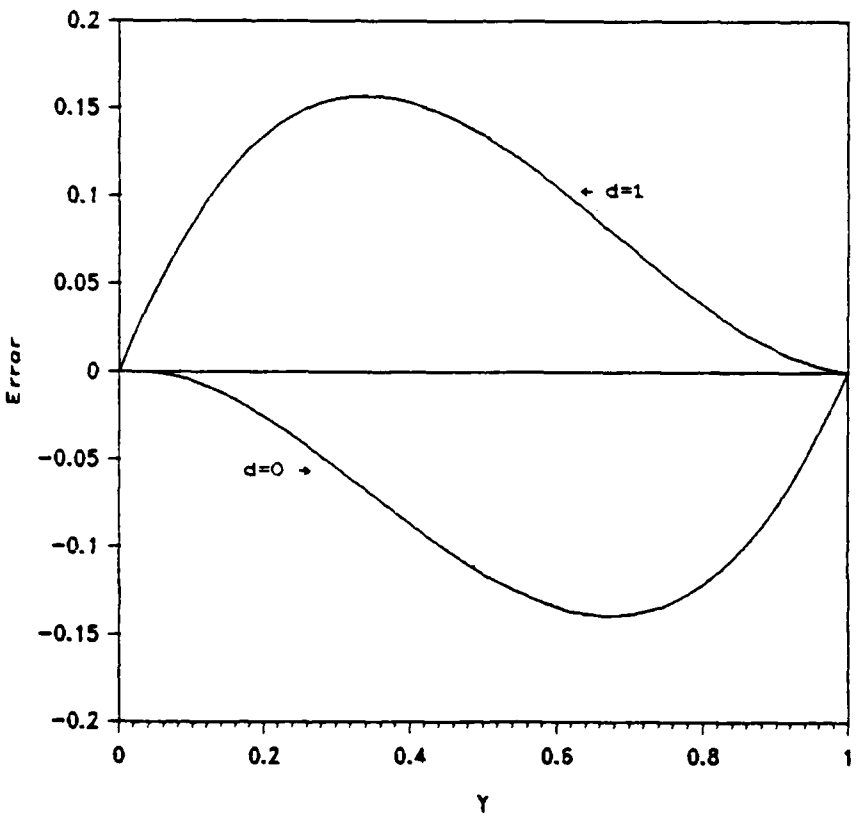


Figure 7. Calculated Error Term.

from zero to one. In either of these two conditions, the term $d_j - y_j$ may provide little or no error information that

can be propagated back down to the hidden layers. Thus, the network will take longer to converge to the correct desired response as the calculated responses move closer to the desired responses.

Modified Training Rule

As seen in Figure 7, a more desirable equation would permit the error to propagate down when there is an actual error. Such an effect can be achieved by using the following equation:

$$\delta_j = y_j^4 - 2\, y_j^2 + d_j \qquad (3)$$

where y_j and d_j are as described in equation (2). Figure 8 shows the calculated error term using equation (3). As

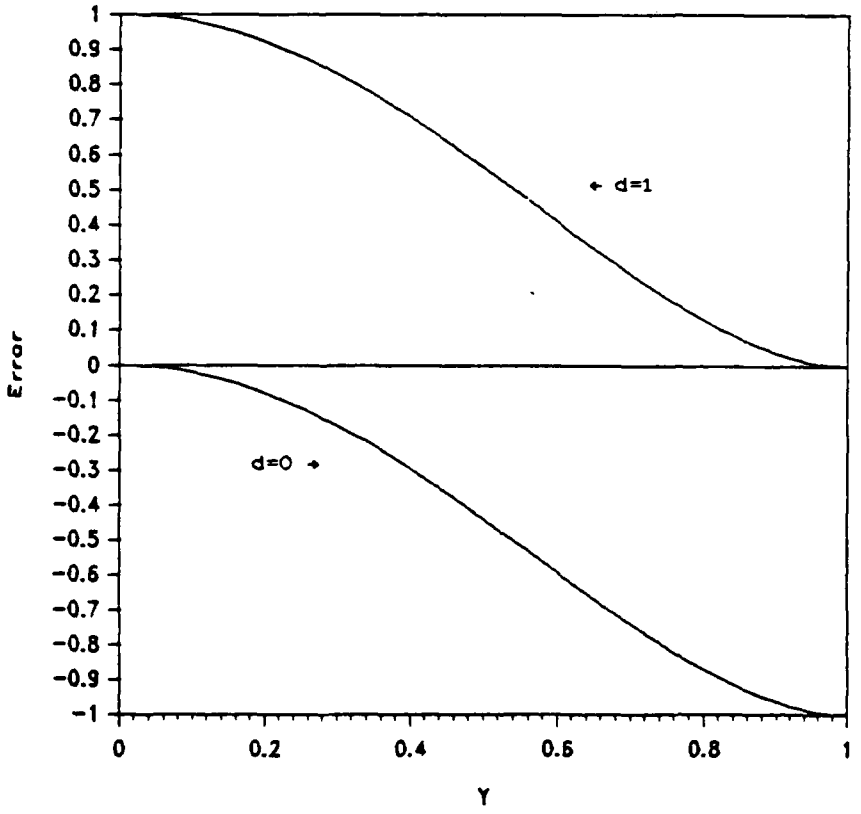


Figure 8. Calculated Error Using Equation (3).

can be seen from Figure 8, when y_j now approaches zero and the d_j is one, the calculated error term goes to zero. Likewise, when y_j approaches zero and d_j equals one, the calculated error goes to zero. By similar observations, it can be seen that when y_j no longer matches d_j, there is an error of one that is of opposite sign. Thus, the full error information is now being propagated downward and the hidden layers are no longer deprived of useful information.

Equation (3) is applied only to the output layer since the desired response (d_j's) is not known in the hidden layers. Further, the remaining steps in the training algorithm stay the same as in the original multi-layer perceptron training algorithm. The reader should consult the article by Lippmann for further information on the training process (5:15).

Testing

Testing consists of three separate tests. In each test, a multi-layer perceptron using equation (3) and a regular multi-layer perceptron are compared. In all cases, the momentum term (see Lippmann (5:17)) is zero.

Test One. In test one, a very simple input data set is used consisting of three different classes. Table I below shows the three "central" vectors that are used to generate the input data.

Table I. Central Data Vectors

Vector	Component					
Class A	1.0	1.0	1.0	-1.0	-1.0	-1.0
Class B	1.0	-1.0	1.0	1.0	-1.0	1.0
Class C	1.0	1.0	-1.0	1.0	-1.0	-1.0

Training consists of selecting one of the three central vectors at random and then adding independently a small random value ranging uniformly from -0.2 to 0.2 to each component of the chosen vector.

Training is stopped after every 20 training iterations and the network is tested. A set of 24 pre-selected test vectors are used and are given in Table II. Notice that the

Table II. Test Vectors

Test#	Component Values						Closest Class
0	1.0	1.0	1.0	-1.0	-1.0	-1.0	A
1	1.0	-1.0	1.0	1.0	-1.0	1.0	B
2	1.0	1.0	-1.0	1.0	-1.0	-1.0	C
3	1.2	1.2	1.2	-1.2	-1.2	-1.2	A
4	0.8	0.8	0.8	-1.2	-1.2	-1.2	A
5	1.2	1.2	1.2	-0.8	-0.8	-0.8	A
6	0.8	0.8	0.8	-0.8	-0.8	-0.8	A
7	1.2	-1.2	1.2	1.2	-1.2	1.2	B
8	0.8	-0.8	0.8	0.8	-0.8	0.8	B
9	1.2	-0.8	1.2	1.2	-0.8	1.2	B
10	0.8	-1.2	0.8	0.8	-1.2	0.8	B
11	1.2	1.2	-0.8	1.2	-0.8	-0.8	C
12	1.2	1.2	-1.2	1.2	-1.2	-1.2	C
13	0.8	0.8	-0.8	0.8	-0.8	-0.8	C
14	0.8	0.8	-1.2	0.8	-1.2	-1.2	C
15	2.0	2.0	2.0	-2.0	-2.0	-2.0	A
16	2.0	-2.0	2.0	2.0	-2.0	2.0	B
17	2.0	2.0	-2.0	2.0	-2.0	-2.0	C
18	0.5	0.5	0.5	-0.5	-0.5	-0.5	A
19	0.5	-0.5	0.5	0.5	-0.5	0.5	B
20	0.5	0.5	-0.5	0.5	-0.5	-0.5	C
21	1.0	0.5	1.0	-0.5	-1.0	-0.5	A
22	1.0	-0.5	1.0	0.5	-1.0	0.5	B
23	1.0	0.5	-1.0	0.5	-1.0	-0.5	C

test vectors 0 to 14 are within the training inputs. Test vectors 15 to 23 are out of the range of the training data, but their Euclidean distances make them closer to one class than to another. When the output node giving the largest response to these vectors matches in class, the output is considered to be correct. Figure 9 shows a plot of the testing scores for each of the two networks as training progresses.

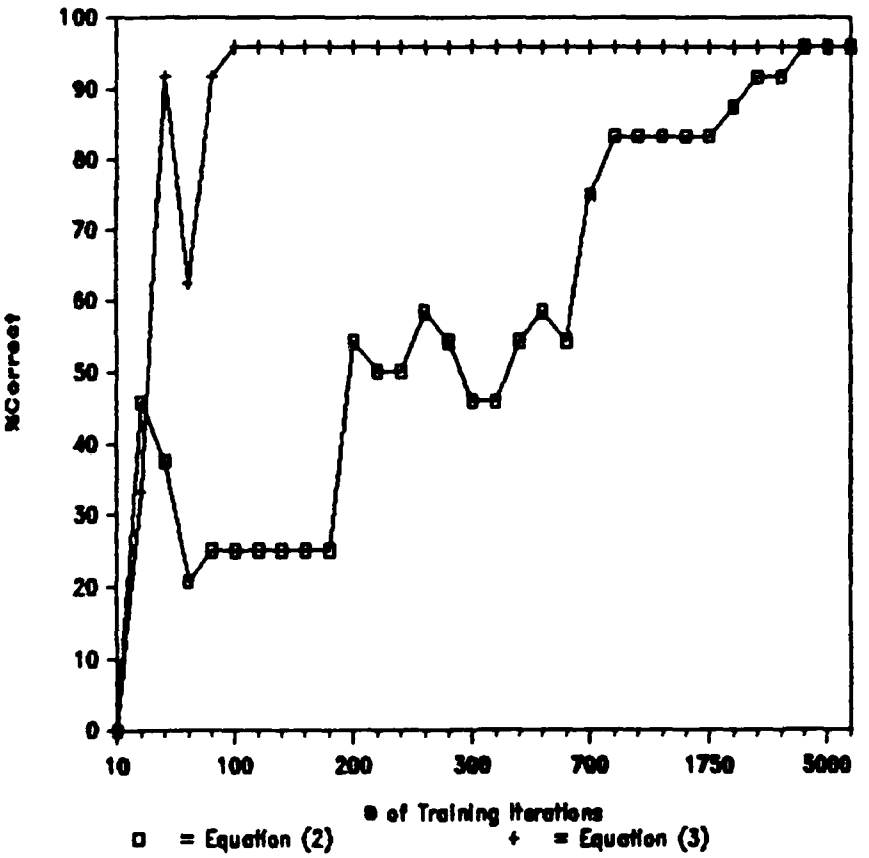


Figure 9. Accuracy Using Table II Test Vectors.

Test Two. In the second part of testing, a more complicated data set is chosen. Again, the data falls within three distinct, uniformly distributed classes and one of the

three classes is disjoint. Figure 10 shows how the data is distributed. As in test one, the nets are identical except that one network is using equation (3).

After every 1000 training iterations, 50 tests are performed to determine each net's accuracy. Testing data consists of randomly choosing a sample point within one of the four distinct areas that compose the three data sets. The correct response requires that the output node match in class and have an output greater than 0.9, while all the

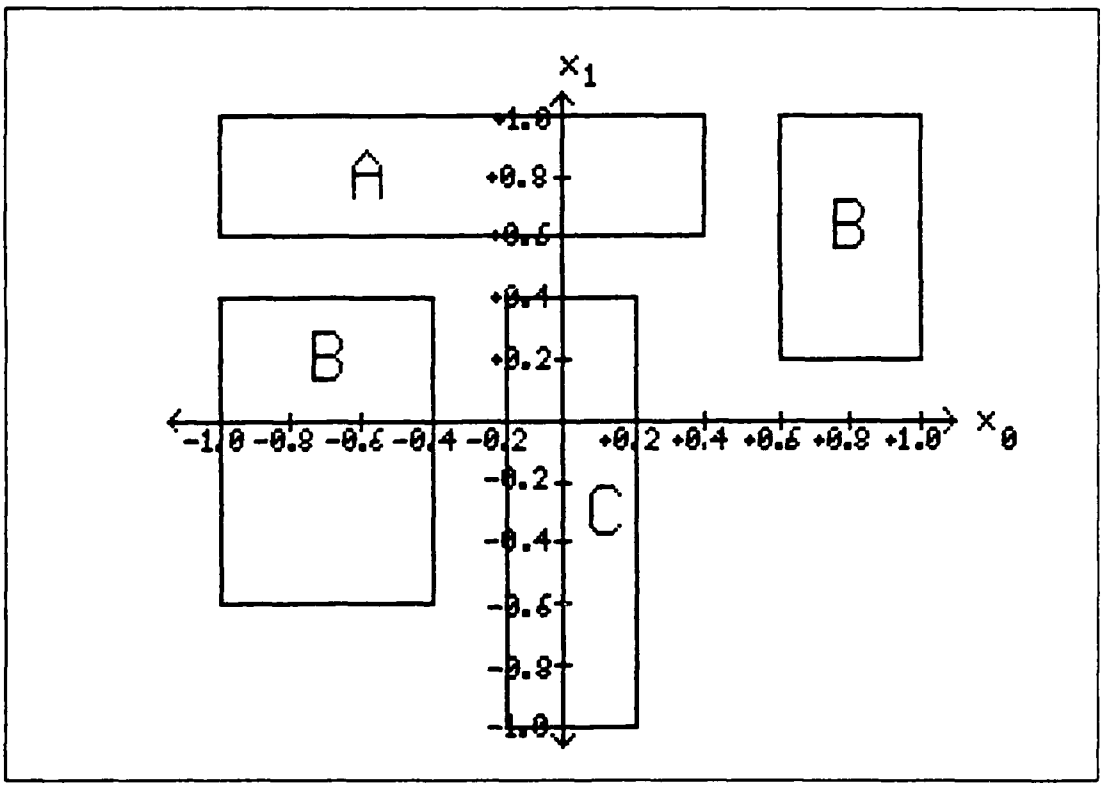


Figure 10. Test Two Data.

other output nodes have an output of 0.1 or lower. Figure 11 shows a plot of the testing scores for the two nets as training progresses.

Test Three. In the third test, the data is from the

thesis work by Capt Dennis Ruck (11) and consists of doppler images collected on 84 observations of tanks, jeeps, trucks, and petroleum oil and lubricant tankers. Twenty-two dimensional, normalized (between zero and one) Zernike moments are then computed and stored in a file (the reader is advised to read the thesis by Capt Ruck for more information pertaining to Zernike moments (11)). As mentioned previously in the scope of this paper, errors in the normalization process were not corrected. Thus, the reader will

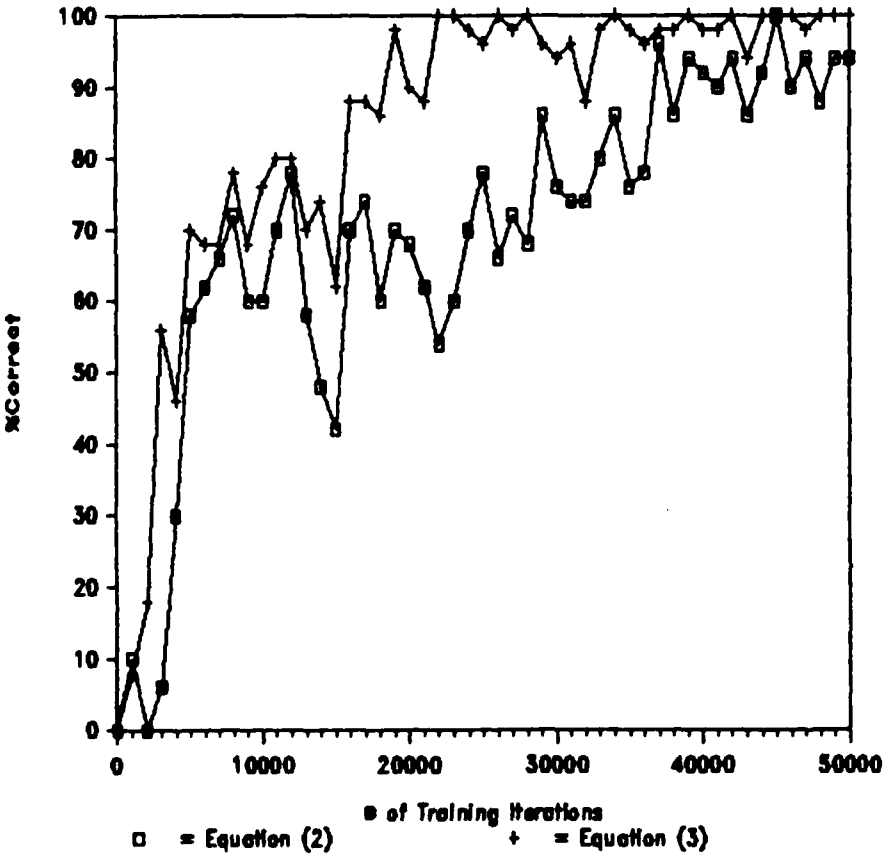


Figure 11. Accuracy For Test Two Data.

notice that no one network achieves 100 per cent correct accuracy in identification for test three.

As before, two identical multi-layer perceptrons, only differing by equation (3), are used. The 84 vectors are divided into 56 training vectors and 28 testing vectors. Training consists of selecting one of the 56 training vectors at random for each of the 50000 training iterations. Testing is then performed after every 500 training iterations and each of the 28 testing vectors are then presented to the network. Figure 12 shows a plot of the testing scores for the two networks as training progresses.

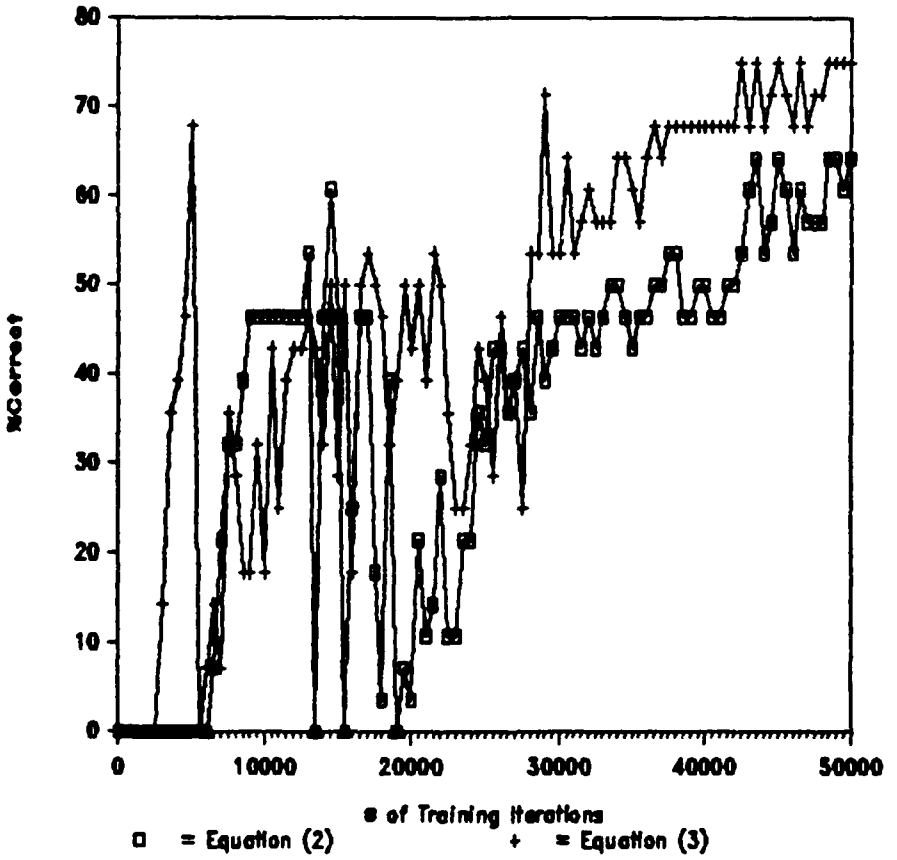


Figure 12. Accuracy For Test Three Data.

Discussion

As can be seen in Figures 9, 11, and 12 the equation (3) does provide a slight improvement in the training performance. Also, there is no additional computational burden; therefore, there is no increase in the run time for this method compared to the regular multi-layer perceptron training time.

It is interesting to note the marked improvement in Test One using the simple data. This result is due to the ease in the testing criteria; only the output neuron with the maximum value had to match in class with the input data. The resulting performance is still significant due to the fact that if a MAXNET (a neural network that is capable of selecting the output node that has the highest output) were used in identifying the output neuron with the maximum output, this same curve would be displayed.

Unexpected Results

Early in the research an unexpected result occurred. According to Lippmann, the thresholds could be trained by using similar update rules while assuming that the thresholds are tied to a constant value (5:17). What was not stated is, what the constant value should be. Therefore, the assumption is made that the constant value of positive one may be used. Testing shows for some cases where the hyperplanes between two classes has infinite or nearly infinite slope that the thresholds have the tendency to go to an

extreme value if a positive one value is used.

As an example of this phenomena, a single-layer perceptron consisting of one neuron is trained for two classes as shown in Figure 13. As can be seen in Figure 13, there is a hyperplane of infinite slope separating the two classes at x_0 equals 0.2. The result of using a threshold tied to a positive one causes the threshold to keep increasing linearly as shown in Figure 14. This in turn causes the output to always be at a value of positive one since the threshold becomes the dominant term in the sigmoid equation. Further, every time the perceptron output is zero, there is

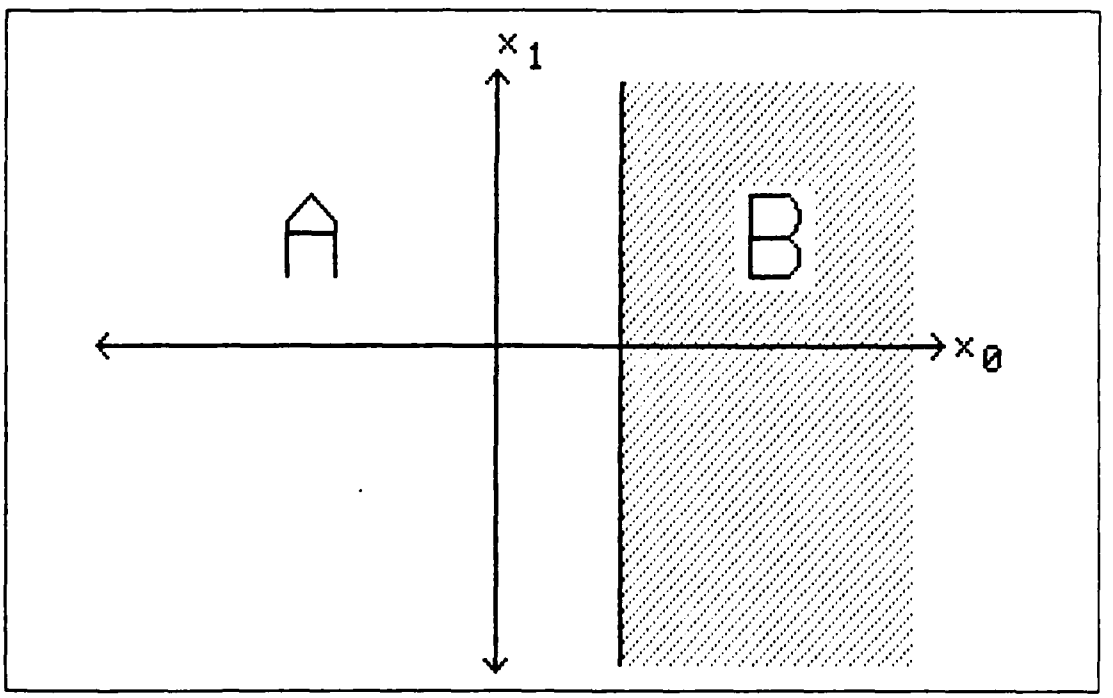


Figure 13. Two Class Problem With Infinite Slope.

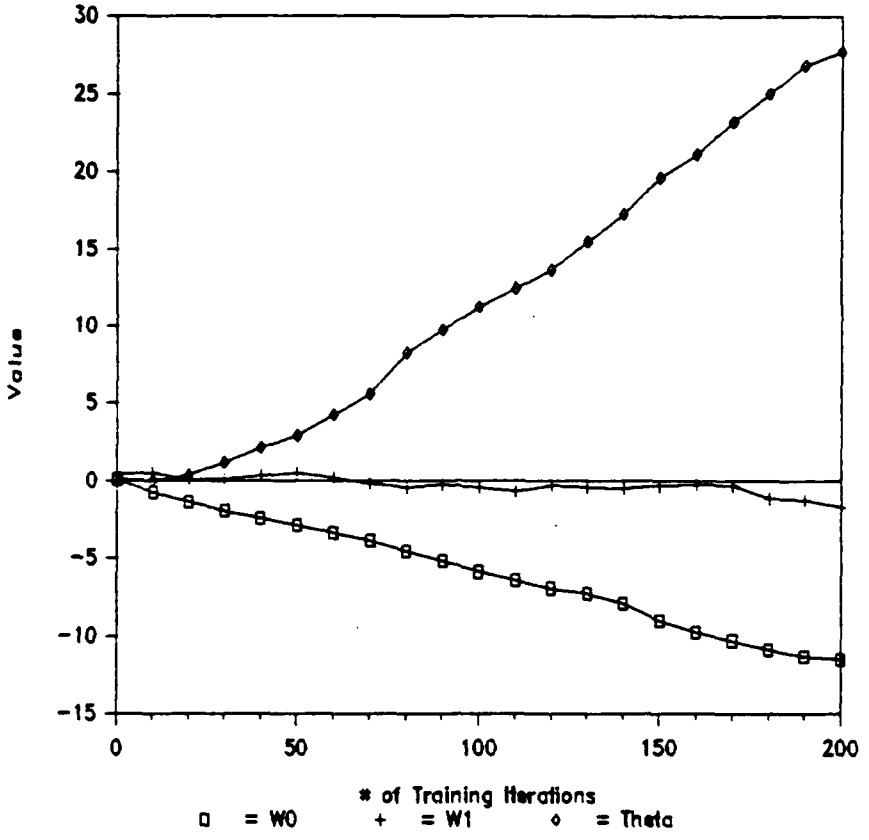


Figure 14. Results With Threshold Tied To +1.

an error and the threshold is increased in value.

If the threshold is tied to a negative one, the problem is resolved and the threshold no longer becomes the dominant term as shown in Figure 15. As can be seen, the weights become dominant and the perceptron trains as expected.

In a multi-layer perceptron, tying the thresholds to a positive one will cause the network to produce only some of the correct answers. Typically, since it is not known if the data can be separated by hyperplanes with near infinite slope, the use of tying the thresholds to a positive one should be avoided especially when the thresholds are to be trained. Thus, it is recommended that the thresholds be

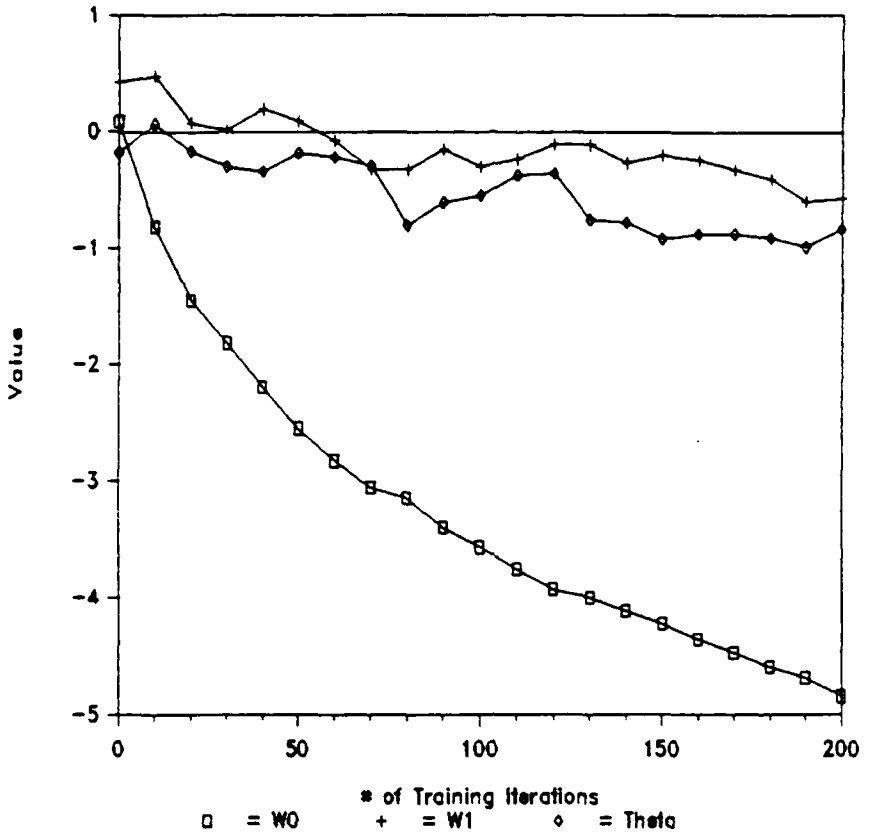


Figure 15. Results With Threshold Tied To -1.

tied to a constant of negative one.

It should be noted that the multi-layer perceptron uses the Generalized Delta Rule developed by Rummelhart (10:123) and is based upon a least means squared approach where the error is (3:142):

$$\text{Error} = 1/2\ (d_j - y_j)^2 \qquad (4)$$

where d_j is the desired output at node j and y_j is the actual output at node j. When the partial derivative of equation (4) is taken with respect to the weights, the threshold term is considered to be a constant. (The reader is advised to see the paper written by Rogers and Stright (8) for more information.) However, if the situation is

reversed and the partial derivative of equation (4) is taken with respect to the threshold and the weights are considered to be at constant values, then the threshold update rule is found to be:

$$\theta(t + 1) = \theta(t) - \eta \, \delta_j \, x'_j \qquad (5)$$

where $\theta(t)$ is the threshold at time t, η is the gain term, δ_j is the error term from equation (2), and x'_i is the input to node j from node i from the layer below. Thus, equation (5) supports the conclusion that the thresholds should be tied to a *negative one* since there is a negative sign in front of the gain (η) term.

New Error Term

Finally, it should be noted that by using the least mean squared approach, equation (3) can be derived from a postulated error term of:

$$\text{Error} = y_j^3 / 3 + y_j^2 / 2 - y_j - \ln(1 - y_j) - d_j \, x'_i \, w_{ij} \qquad (6)$$

where y_j, d_j, x'_i, and w_{ij} are defined as before in equations (1) and (2). It is interesting to note as y_{ij} approaches one, the error goes to negative infinity. No explanation can be given for the meaning of equation (6) other than it enables the derivation of a better response to errors.

III. Accelerated Learning for the Multi-Layer Perceptron By Expanding the Sigmoid Function

Introduction

The purpose of this chapter is to explore decreasing the training time for a multi-layer perceptron. The first section briefly discusses the sigmoid function and the effect of increasing the rate at which the sigmoid rises. In the second section a test is performed showing the effects of increasing the rate of the sigmoid. The third section introduces a new accelerated learning method based on the increased rate of the sigmoid. The fourth section describes the testing that is performed. The last section discusses the results and the need for more testing in this area.

Increasing the Rate

Several attempts have been made at reducing the training time required for multi-layer perceptrons. For example, Shepanski (12) and Dahl (2) both have derived methods to decrease the training time. However, their methods increase the computational burden and in some cases cannot be applied directly to a general form of a multi-layered perceptron. However, it has been noted by Yang and Guest (14:369) that by increasing the rate at which the sigmoid rises, the training time is decreased without an increase in the number of computations that need to be performed.

In general, the sigmoid function has the form:

$$y = (\epsilon \; / \; (1.0 + \exp(-\beta * \alpha + \phi)) - \mu \qquad (7)$$

where ϵ (typically one) is the amplitude, β (typically one) is the rate at which the sigmoid rises, α for neural networks is the weighted sum of the inputs minus the threshold, ϕ (typically zero) allows the sigmoid to shift left or right on the α axis, and μ (typically zero) allows the sigmoid to shift up or down on the y axis. By increasing β, the $\beta * \alpha$ surface between classes begins to sharpen. Thus, a larger error term can be calculated even when the two classes are close to each other. Therefore, it seems reasonable to increase β for the multi-layered perceptron since better error information can be propagated down through the net.

Rate (β) Testing

To see if the rate (β) increase can decrease the training time, a simple test is performed. The data is uniformly distributed among three classes. One of the classes is a disjoint region as shown in Figure 16.

A three layer multi-layer perceptron (with 2 inputs, 12 neurons in the first layer, 12 neurons in second layer, and 3 output neurons) is used and after 20000 training iterations a test is performed. Fifty test vectors are chosen at random from one of the four data regions. Scoring is computed based on requiring a 0.9 level or better for the correct output, while all other outputs must be at 0.1 or lower. Figure 17 shows that as the rate increases, perfor-

mance decreases (except for a rate of 10). One would expect

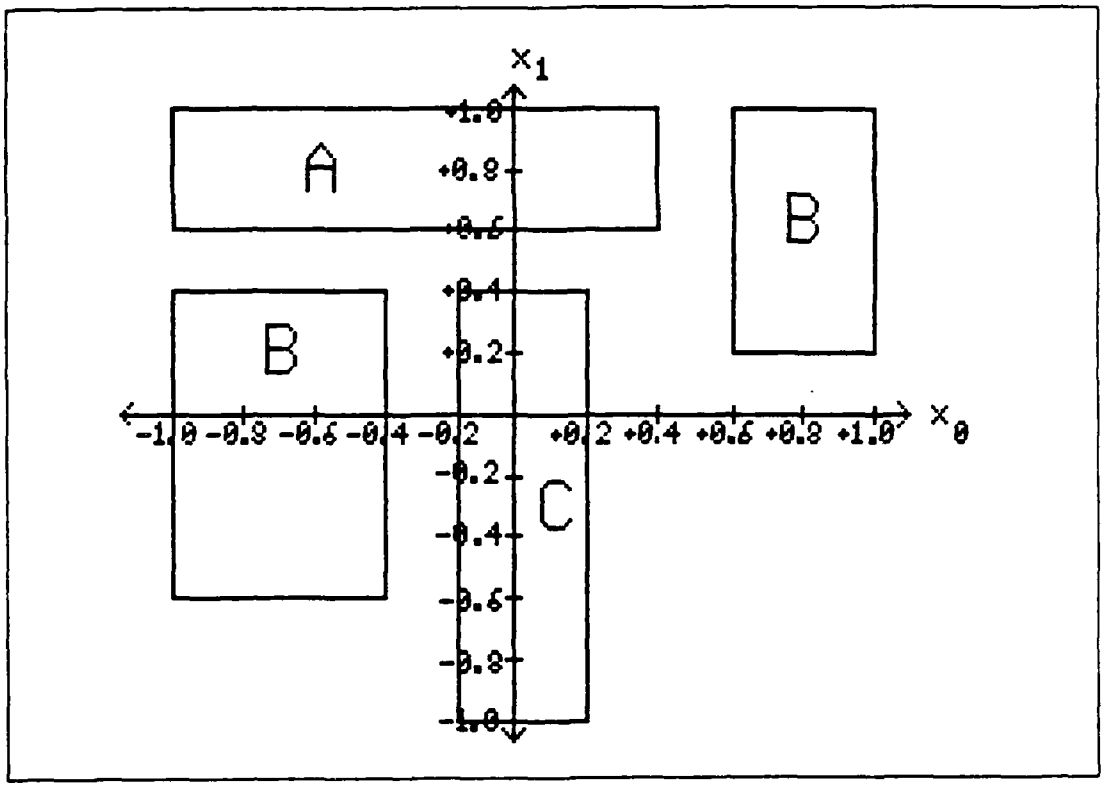


Figure 16. Rate (β) Testing Data.

that the curve should go to 100 per cent and stay there if an increase in rate improves the training speed. Instead, Figure 17 shows that the network becomes unstable (that is, accuracy rolls off) as the rate increases beyond the value of 10.

New Training Rules

The reason the network becomes unstable is due to the fact that the original training rules do not explicitly account for an increase in β. As mentioned previously, the multi-layer perceptron uses the equation:

$$\text{Error} = 1/2\ [d_j(t) - y_j(t)]^2 \qquad (8)$$

where d_j is the desired output at node j and $y_j(t)$ is the

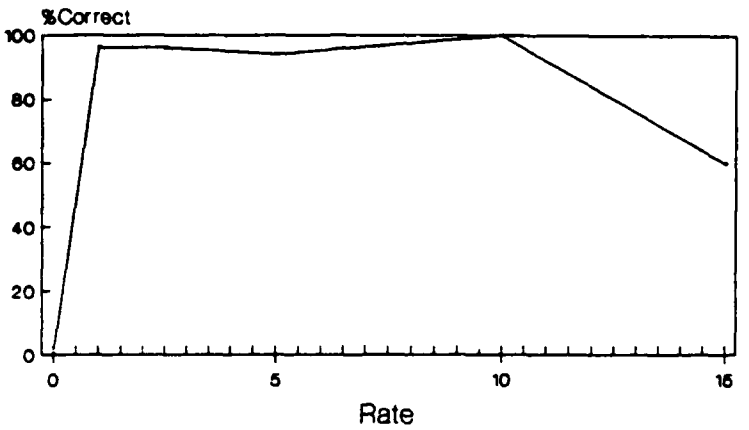


Figure 17. Performance Versus Rate After 20000 Training Iterations.

calculated output at node j at time t. It can be shown by using equations (7) and (8) that the new training rule for the output of the multi-layer perceptron becomes:

$$w_{ij}(t + 1) = w_{ij}(t) - (\eta\beta/\epsilon) [y_j(t) + \phi] [y_j(t) + \phi - \epsilon] x'_i \quad (9)$$

where w_{ij} is the weight from node i in the hidden layer below to node j in the output layer, η is the gain term, ϵ is the amplitude of the sigmoid, ϕ is the shift term in equation (7), and x'_i is the input to node j from node i in the hidden layer. The threshold training for the output node j becomes:

$$\theta_j(t + 1) = \theta_j(t) + (\eta\beta/\epsilon) [y_j(t) + \phi] [y_j(t) + \phi - \epsilon] \quad (10)$$

where the terms are the same as in equation (9) and $\theta_j(t)$ is the threshold at time t.

For the hidden layers, the training rule for the weights becomes:

$$w_{ij}(t+1) = w_{ij}(t) - (\eta\beta/\epsilon)\ [y_j(t) + \phi]\ [y_j(t) + \phi - \epsilon]\ (\sum_k \delta_k\ w_{ij})\ x'_i \qquad (11)$$

where w_{ij} is the weight from either the hidden layer node i below or from the input i to node j, δ_k is the error at node k in the layer above, x'_i is the input to node j from either node i in the hidden layer below or the input i, and the rest of the terms are those previously mentioned. Likewise, the threshold training rule for the hidden node j becomes:

$$\theta_j(t+1) = \theta_j(t) + (\eta\beta/\epsilon)\ [y_j(t) + \phi]\ [y_j(t) + \phi - \epsilon]\ (\sum_k \delta_k\ w_{jk}) \qquad (12)$$

where the terms are those previously mentioned. Note, that when ϵ equals one, β equals one, and ϕ equals zero, the equations (9) to (12) reduce to the original multi-layer

Testing

Testing consists of two separate tests. In the first test, a multi-layer perceptron of identical size uses the same data as previously mentioned. In the test, different rates are again used to see if the new equations will provide stability.

In the second test, a multi-layer perceptron uses the Ruck (11) data introduced in the previous chapter. Two different rates are tried for the Ruck (11) data. Also, as in chapter two, no attempt is made at fixing the testing data.

Test One. Testing consists of three uniformly distributed data sets that were originally shown in Figure 16. A multi-layer perceptron is constructed of the same size as in the original rate testing but now employing equations (9) to (12). Also, ϵ and ϕ are kept to their original values of one and zero respectively. Thus, it should become clear whether or not any improvement using the new equations and increasing the sigmoid is possible or not.

Six different β terms are chosen (1.0, 2.5, 5.0, 7.5, 10.0, and 15.0) for testing. Training is stopped after 1000, 2000, 5000 and 20000 iterations. At each of these stopping points, 50 tests are performed and the score is noted as before in the original rate testing.

Test Two. In test two, a multi-layer perceptron of the size described in Chapter One is used. That is, there are 22 inputs, 20 neurons in the first layer, 6 neurons in the second layer, and 4 output neurons. The data is as before: 56 training vectors and 28 testing vectors. First, the multi-layer perceptron will train using a rate (β) equal to five. As in Chapter One, a total of 50000 training iterations are to be used. Testing is done after every 500 training iterations. Finally, a new multi-layer perceptron of the same size is used with a new rate of ten. Training and testing is conducted in the same manner.

Results and Discussion

Figures 18 through 21 show the performance of the test

one training. After 1000 (see Figure 18) training iterations the multi-layer perceptron is beginning to respond to the increased rate of the sigmoid.

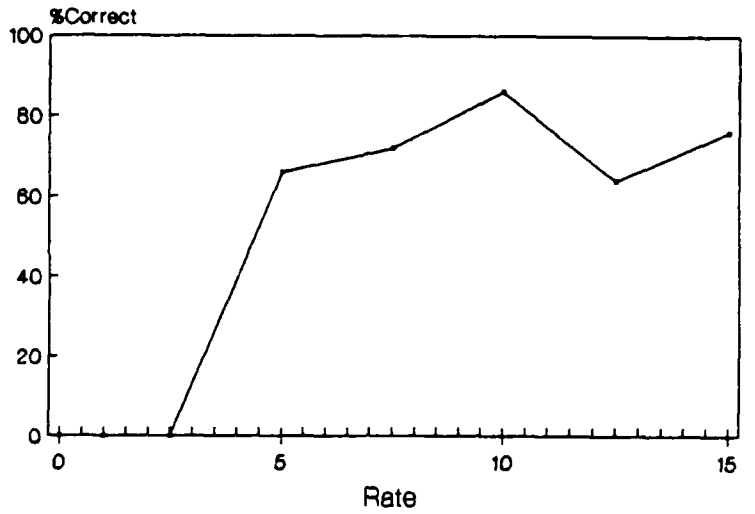


Figure 18. Accuracy After 1000 Training Iterations.

After 2000 training iterations (Figure 19), the networks with the rates of 10.0 and 15.0 are already scoring 100 per cent while a regular multi-layer perceptron (that is a mul-

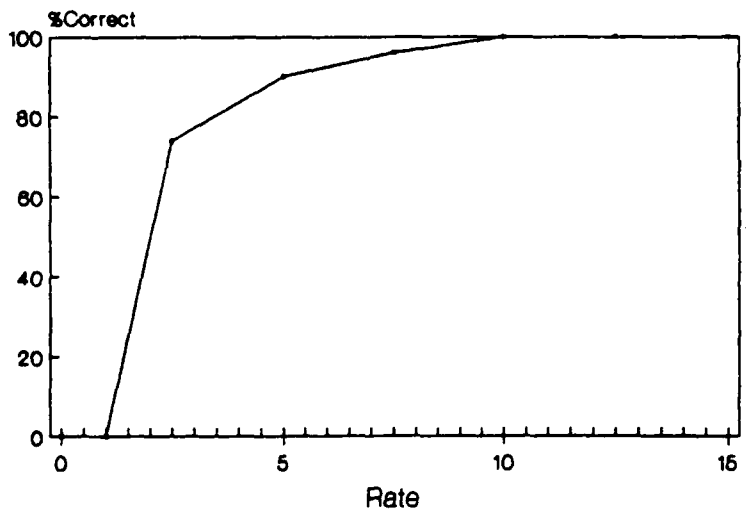


Figure 19. Accuracy After 2000 Training Iterations.

ti-layer perceptron where the rates of the sigmoids are equal to one) is still scoring 0 per cent.

By 5000 training iterations (Figure 20), the networks with rates greater than 5.0 are at 100 per cent with no sign of instability at the higher rates. Also note that the multi-layer perceptron with a rate of one is beginning to respond to the training.

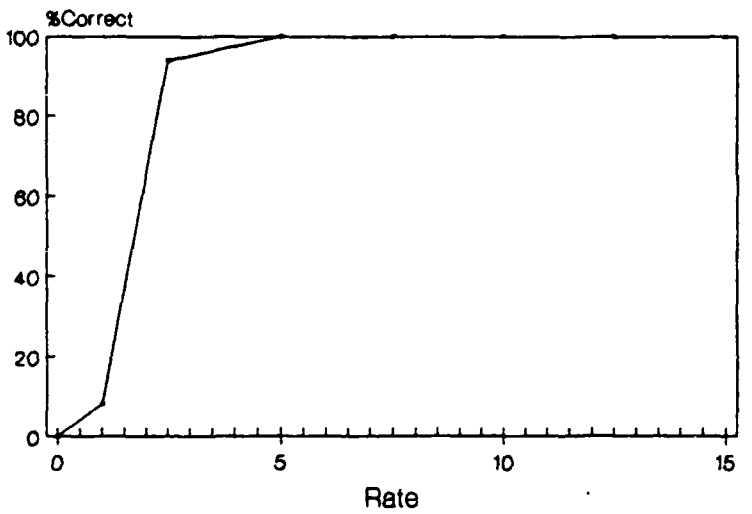


Figure 20. Accuracy After 5000 Training Iterations.

Finally, after 20000 training iterations (Figure 21) all the networks with a rate greater than 1.0 are scoring 100 per cent. In comparison with Figure 17, there are no signs of instability as the rate increases. The figures show that training time is decreased by an order of magnitude if using a rate of 10.0 or 15.0.

Figures 22 and 23 show the performance of the test two training. As observed in Figure 22, the multi-layer perceptron, using a rate of 5.0, is doing slightly better than the

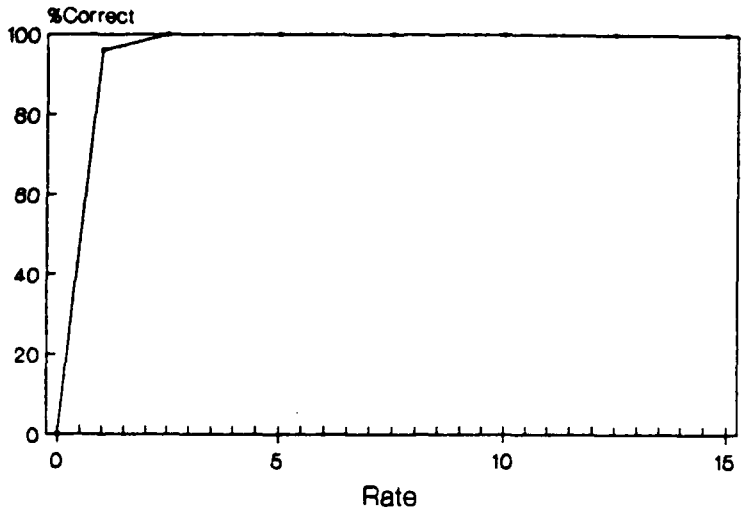


Figure 21. Accuracy After 20000 Training Iterations.

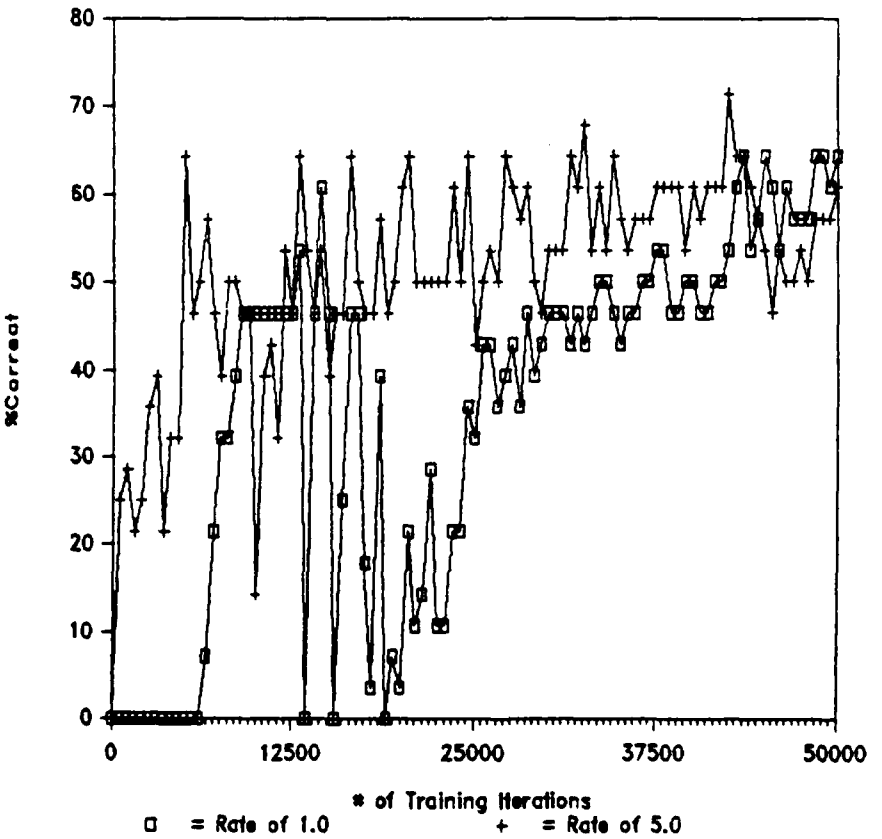


Figure 22. Performance for Test Two with a Rate of 5.0.

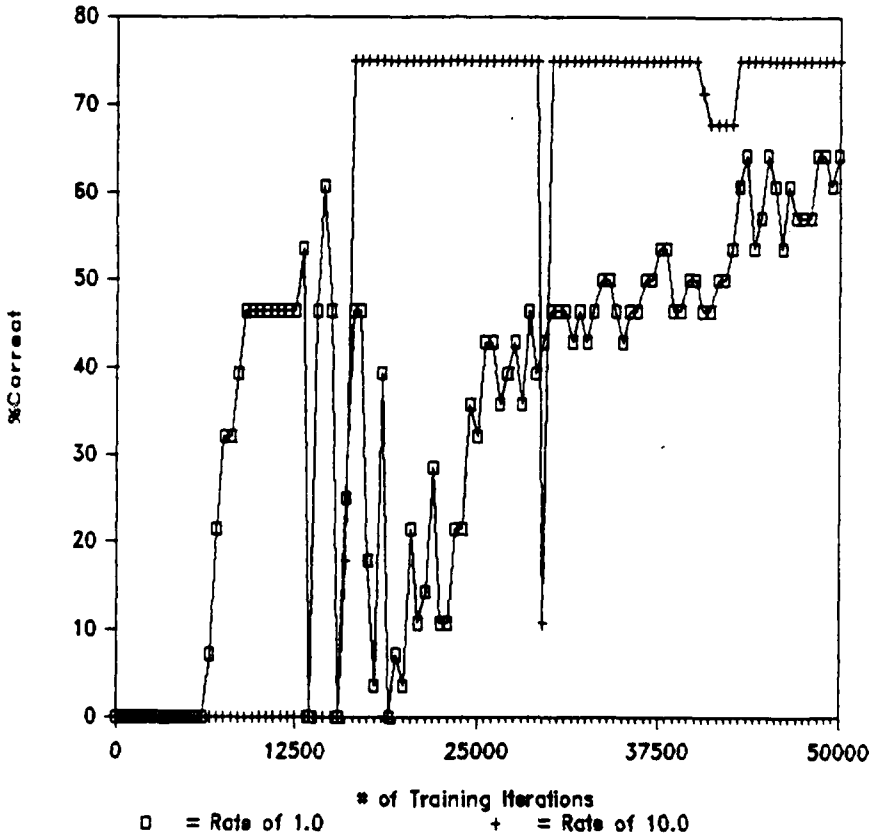


Figure 23. Performance for Test Two with a Rate of 10.0.

regular multi-layer perceptron. However, as can be seen in Figure 23, when the rate is 10.0 the performance jumps up to 75 per cent after 21000 training iterations. It is not known why the multi-layer perceptron does not respond to training until after 21000 iterations when the rate is at ten. But overall, the networks did as well or better than the regular multi-layer perceptron without increasing the number of computations.

Though the results are promising, other preliminary tests indicate that if ϵ is set to two and ϕ is set to one (the outputs of the neurons are now negative one to positive one), the network becomes unstable as the rate (β) increases

above one. Therefore, more testing and investigating will be required to fully determined what the true error equation should be in order to permit an increase in all factors, or whether there is a limit to the type of sigmoid curve that can be used in training. However, if the outputs are kept between to zero and one, increases in the rates are allowed. This method can be used without additional computational burdening and an increase in training speed is therefore possible.

IV. Additional Class Training for a Multi-Layer Perceptron

Introduction

The purpose of this chapter is to discuss whether a partially trained multi-layer perceptron can train faster than an untrained multi-layer perceptron when a new class is to be added to the problem space. The first section discusses the background of adding a new class. The second section describes the testing that is to be performed. The final section presents the results of testing and describes the effect of adding a new class to a previously trained multi-layer perceptron.

Background

Typically, a multi-layer perceptron is trained with all the classes it will ever need to identify. It has been suggested that perhaps when a new class is to be added, the training time for a multi-layer perceptron could be reduced by using an previously trained network and adding new neurons for additional class discrimination (9). This seems reasonable since most of the network is already trained for the previous classes and therefore the only training that is needed is for those new neurons that are added to the network.

Testing

To test this suggestion, three separate tests are performed using two dimensional data so that the movement of hyperplanes can be easily understood. However, if higher dimensions of data were used, the results would be similar since the training rules for the multi-layer perceptron will move all hyperplanes in the same manner.

The multi-layer perceptrons are initially trained for two classes. In the first test, a complicated data region is added and multi-layer perceptrons of various sizes are tried to determine when the network will train to discriminate all three classes. In test two, a simple data region is added and again multi-layer perceptrons of various sizes are tested to determine when the network will train to discriminate all three classes. Finally, in test three, both the more complicated and the simple data regions are used, but in each case only the newly added neurons will undergo training.

Test One. Initially, a multi-layer perceptron is trained to distinguish between two classes (A and B) with uniformly distributed data. Class B is disjoint as shown in Figure 24. Using a multi-layer perceptron consisting of 6 neurons in the first hidden layer, 6 neurons in the second hidden layer, and 2 neurons in the output layer; 100 per cent correct identification is achieved after 50000 training iterations. (Scoring is based on 0.9 or better for the correct class while the other output is at 0.1 or lower).

Next, a new class (C) is introduced that also has a uniform distribution and is disjoint as shown is Figure 25.

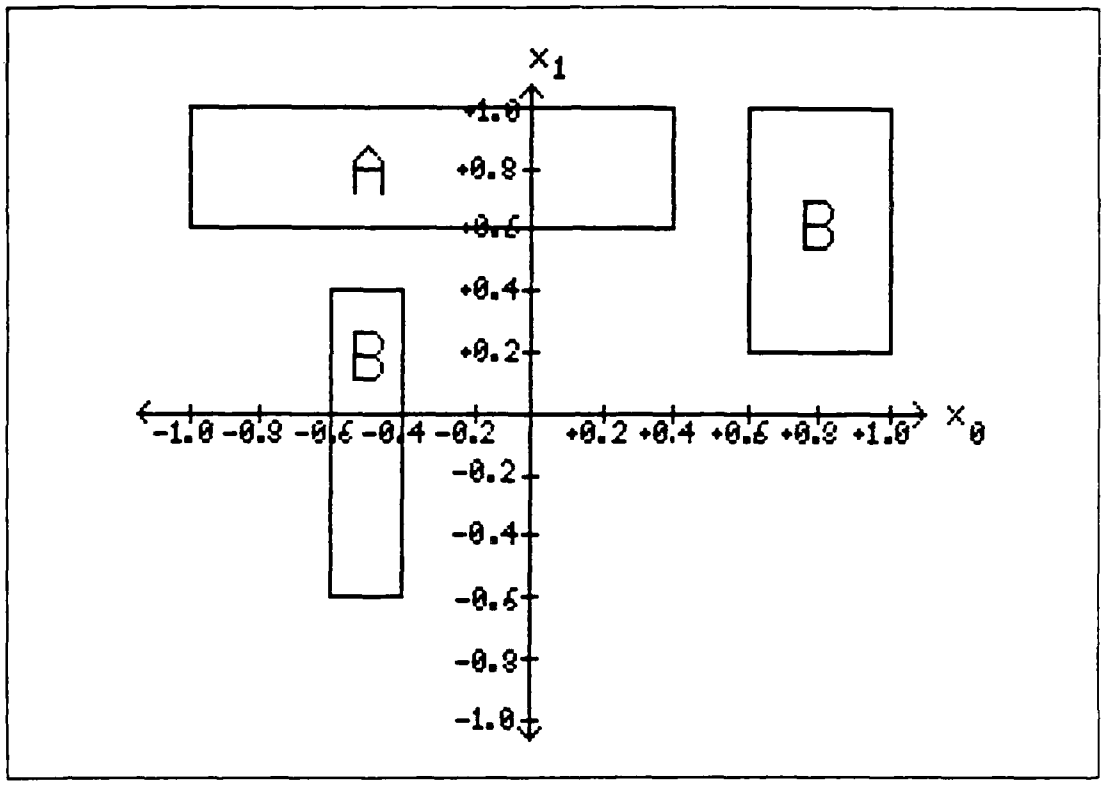


Figure 24. Original Training Data.

As mentioned, new neurons are added to each layer and the network is retrained using random samples from each class. Data from each class is used to prevent the network from unlearning the two previously trained classes. Training is stopped after every 1000 steps and testing is performed to determine the relationship of accuracy versus the required training time to add a new class. Training is stopped after 50000 training iterations and the network size is increased. In each case, a new network matching this size increase is trained for comparison.

Test Two. As in test one, a multi-layer perceptron

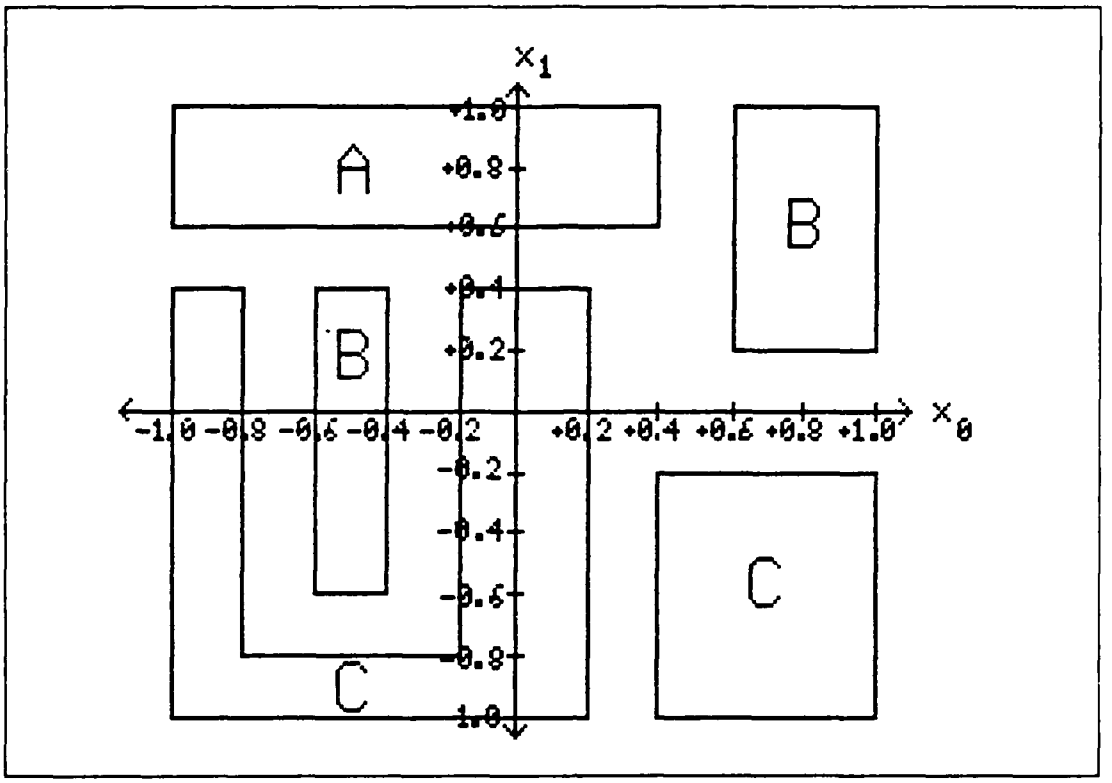


Figure 25. Introduction of a New Class For Test One.

consisting of 6 neurons in the first layer, 6 neurons in the second layer, and 2 neurons in the output layer is trained to achieved 100 per cent accuracy for the data shown in Figure 24 after 50000 training iterations. In this test however, the new class C to be added is the region shown in Figure 26. As before, neurons are added, and training is stopped after every 1000 training iterations, and the network is tested to determine accuracy. Again, training is compared to a new network of the same size.

Test Three. In test three, the multi-layer perceptron consisting of 6 neurons in the first layer, 6 neurons in the second layer, and 2 neurons in the output layer is trained for two classes (A and B) and a 100 per cent accuracy is

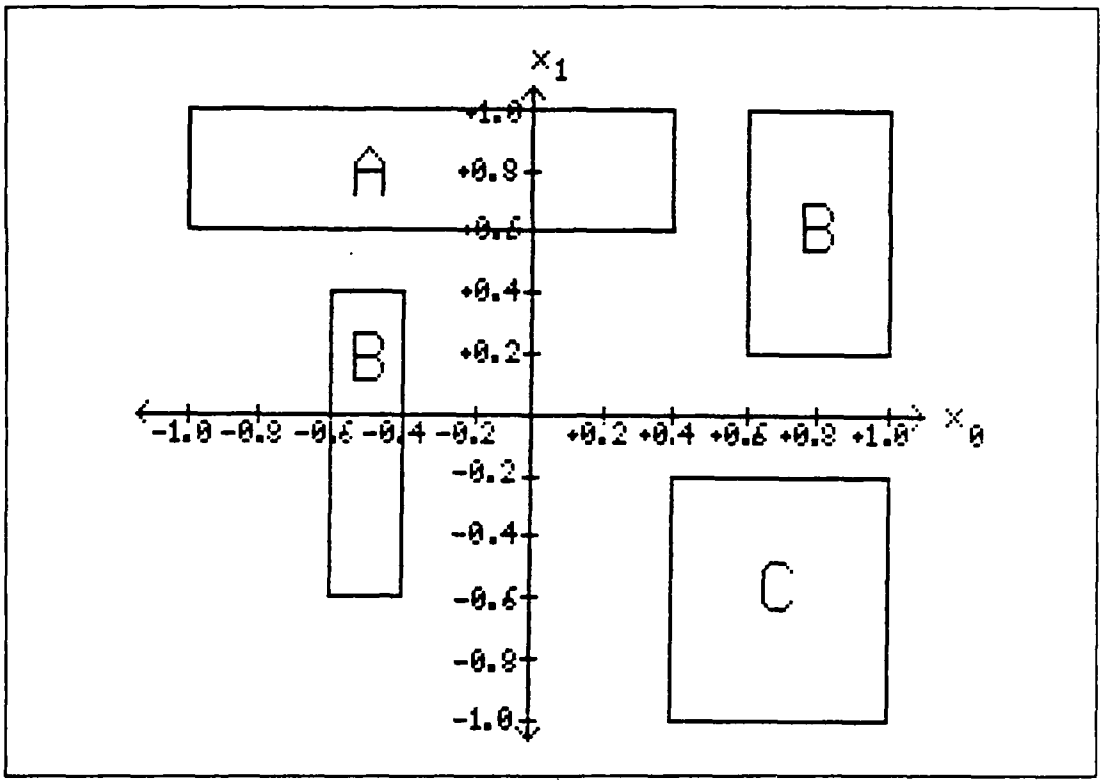


Figure 26. Introduction of a New Class for Test Two.

achieved after 50000 iterations. There are two parts to this test. First, the data region added (C) is that from test one as shown in Figure 25 and only the new neurons that are added to accommodate the new class are to be trained. In the second part of this test, the data region added is that from test two as shown in Figure 26. Again, only the newly added neurons are trained. Testing is stopped as before after 1000 training iterations and the network is tested for accuracy. All results are to be compared to new networks of the same size.

It should be noted in both parts of this test, that because only newly added neurons undergo training, the connections from the new neurons to the old neurons are set to

zero and not permitted to undergo training. Whereas, the new connections from the old neurons to the new neurons are randomly set and permitted to undergo training. If this procedure is not followed, the old neurons will be receiving new data from the connections made from the new neurons. Thus, the old neurons would be required to update their weights, or at least their thresholds, because new information is being received.

Results and Discussion

Figures 27 through 31 shows the scoring accuracy of a partially trained network versus a newly trained network as the net size grows for test one training. (Note: in the following figures, the "Net Size" is as follows: number of inputs - number of first layer neurons - number of second layer neurons - output neurons). As can be seen, there is

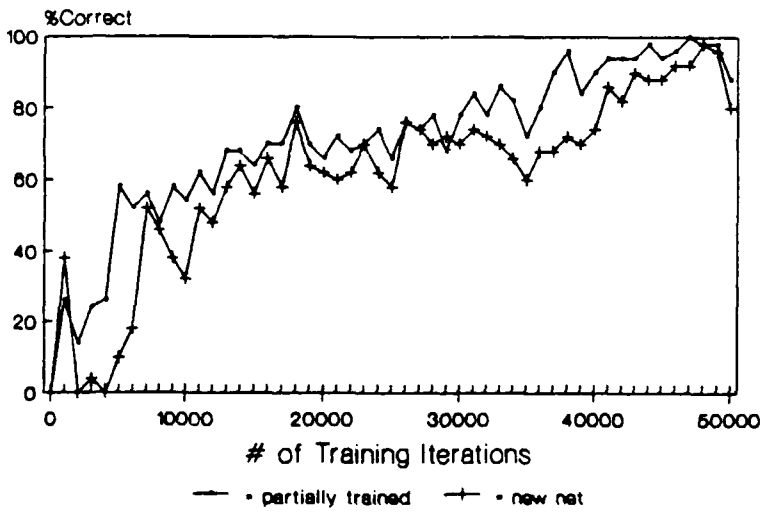


Figure 27. Accuracy for a Multi-Layer Perceptron Using a Net Size of 2-10-8-3 for Test One Data.

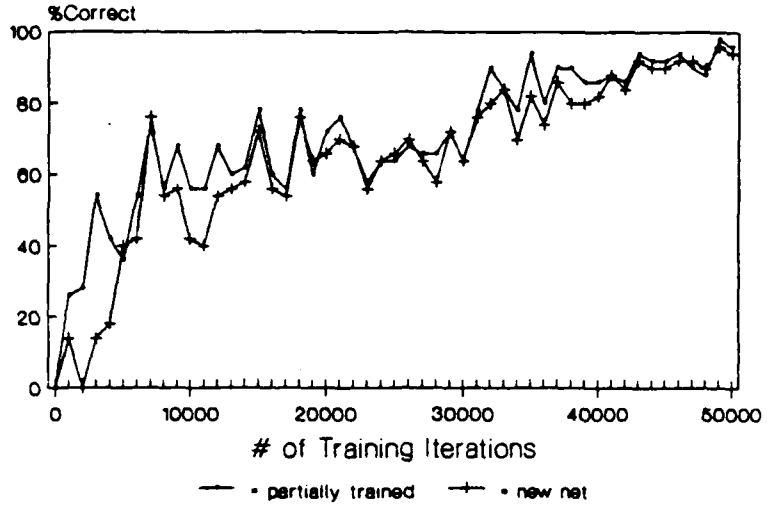


Figure 28. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-16-12-3 for Test One Data.

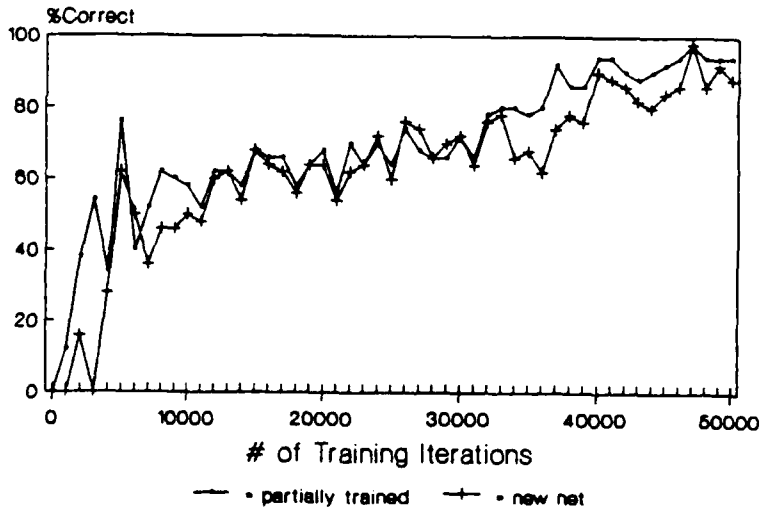


Figure 29. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-21-14-3 for Test One Data.

only a slight advantage when a partially trained network is used because some of the hyperplanes will have to move when a new data region is added. Thus, the hyperplanes in both the partially trained network and the new network can be considered in random positions and will take about the same time to train.

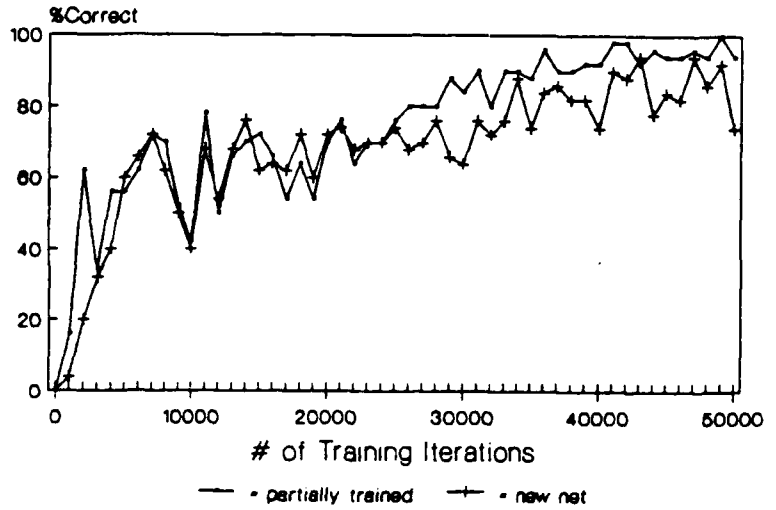


Figure 30. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-30-16-3 for Test One Data.

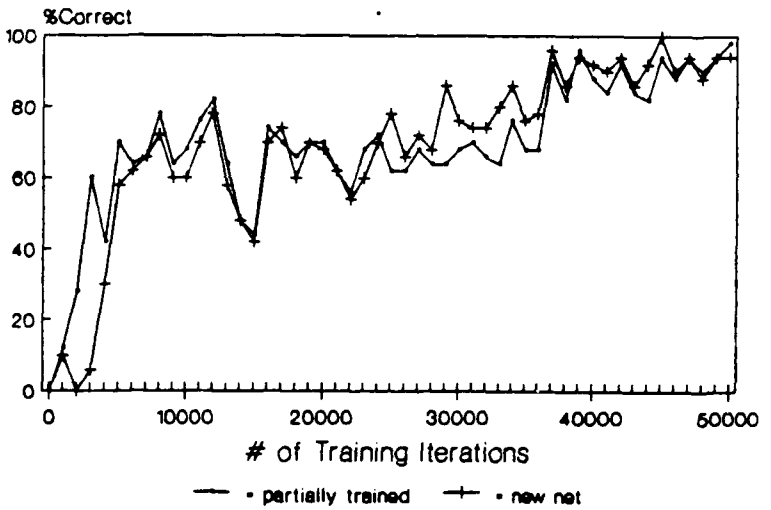


Figure 31. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-36-20-3 for Test One Data.

Figures 32 and 33 show the results of test two training. From these figures it can be seen that the partially trained network does not train as might be expected. Thus, there is no advantage to using a partially trained network. In this case, the newly added data does not fall between the old data regions as in test one. Therefore, the hyperplanes

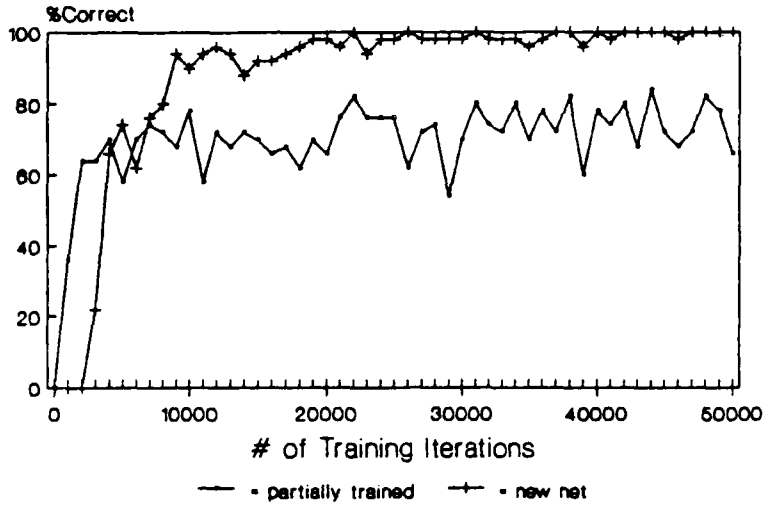


Figure 32. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-12-12-3 for Test Two Data.

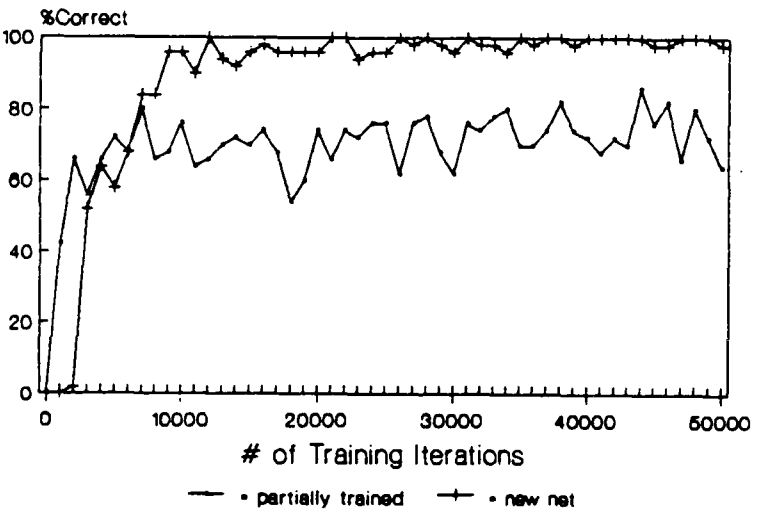


Figure 33. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-16-16-3 for Test Two Data.

will be in the correct position when old data is selected and in the wrong position when new data is selected. Thus, a cycle develops moving the hyperplanes toward old data when old data is selected and away when new data is selected.

Figures 34 and 35 show the results of the test three training with the simple data region C. Figures 36 and 37

show the results of the test three training with the more complicated data region C. Even though training is performed only on the newly added neurons, that is, only the new neurons are allowed to adjust their weights and thresholds, it can be seen from Figures 34 through 37 that no progress is being made by the partially trained networks.

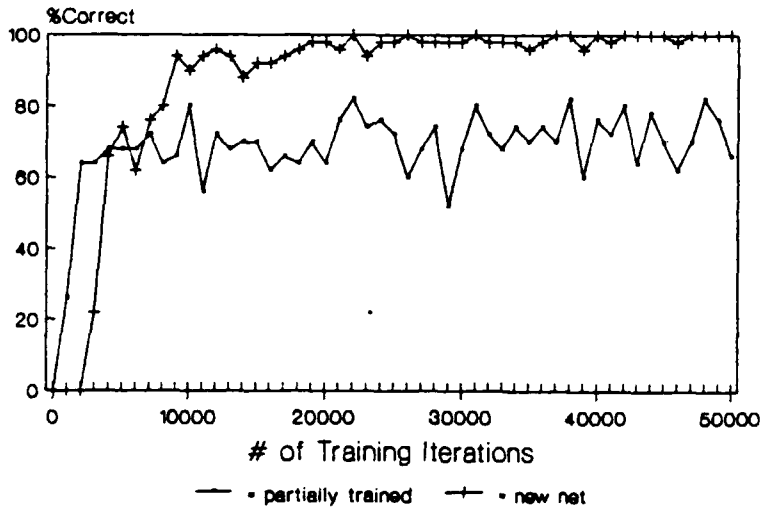


Figure 34. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-12-12-3 for Test Three.

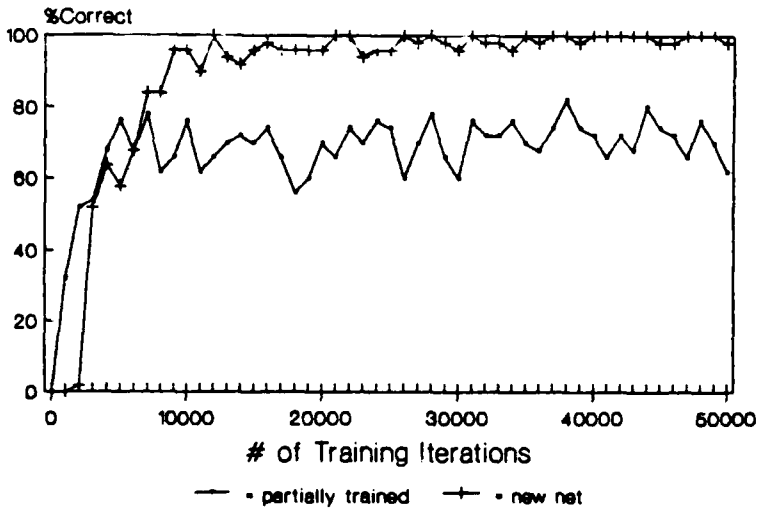


Figure 35. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-16-16-3 for Test Three.

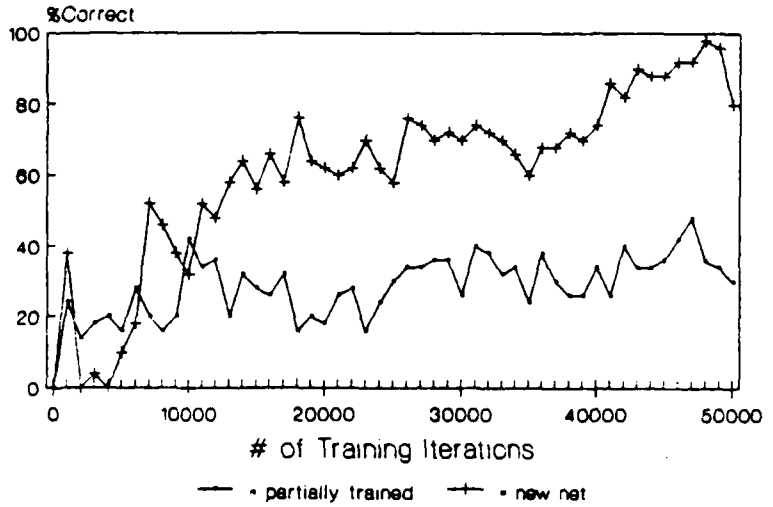


Figure 36. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-10-8-3 for Test Three.

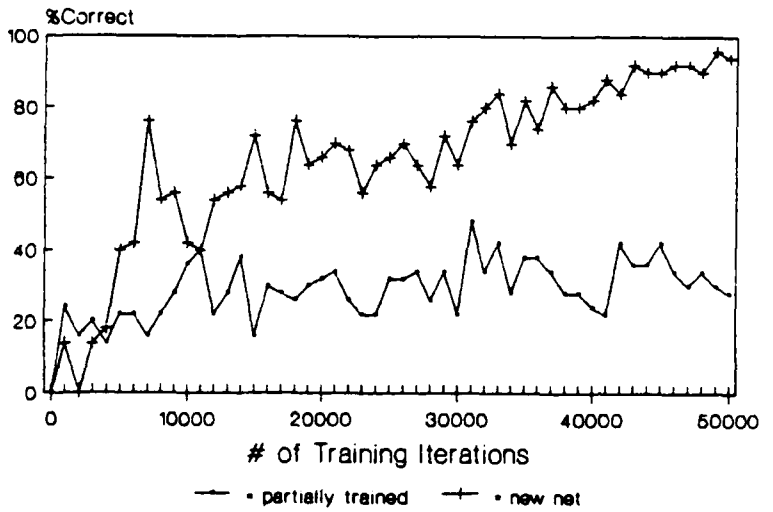


Figure 37. Accuracy for a Multi-layer Perceptron Using a Net Size of 2-16-12-3 for Test Two Data.

In both cases (simple and complicated data regions) the old hyperplanes are not allowed to move at all, since their weights and thresholds did not undergo training. Therefore, when the simpler data region is used and data is selected from this region, the old hyperplanes produce an output for class B. Likewise, there is also an output for class C

produced by the new hyperplanes.

This effect becomes even worse when the more complicated data region is used, since there is an increase in probability that data will fall on one side or the other of the older class B hyperplanes. Only when class A data region is selected is the output from the network correct.

In conclusion, it is recommended that multi-layer perceptrons be reset to the usual random initial condition state when a new data class is to be added, since it is generally not known whether the data region to be added will fall between the older data regions. Even if it were known, there is still only a slight advantage to using a partially trained network.

V. Noise Reduction Using A Multi-Layer Perceptron

Introduction

The purpose of this chapter is to discuss whether a multi-layer perceptron can be trained to reduce noise in a signal. The first section briefly discusses the background of using a multi-layer perceptron as a noise reduction device. The second section describes the testing to be performed. The third section presents the results of testing and recommendations.

Background

Tamura and Waibul suggest that the multi-layer perceptron can be used to reduce noise from speech (13). They contend that a multi-layer perceptron should be able to map noisy speech signals to noise-free signals since "An arbitrary decision surface can be formed ..." (13:553). In their work they state that the noise reduction is "comparable to or better than the conventional power spectrum subtraction method." (13:556). This seems reasonable since a multi-layer perceptron should be able to find the mapping necessary to act as a filter.

Testing

Testing is performed in two phases. In the first phase, the signal source consists of three sine waves at various frequencies that are added together. Then uniform pseudo-

random noise is added and the network is trained to remove the noise from the signal. In the second phase, speech sampled at 8 KHz is used as the signal source. Again uniform pseudo-random noise is added and the network is trained to remove the noise.

Phase One Testing. In phase one, the signal source is created by adding three sine waves together. The three frequencies chosen are 100 Hz, 500 Hz, and 1000 Hz. In addition, each of the three sine waves are allowed to vary independently in phase over the range of $-\pi$ to $+\pi$ radians. The summed sine waves are then sampled at a 8 Khz rate for a total of 60 samples. Uniform pseudo-random noise, varying from negative three to positive three, is added to each of the 60 samples. The amount added is independent from sample to sample. The resulting signal-to-noise ratio is about 0 db.

After the 60 samples are presented for training, 60 new samples are produced by adding three sine waves at the original frequencies but with newly chosen random phases. Again uniform noise varying from negative three to positive three is added to each of the 60 samples.

Since a multi-layer perceptron can only produce an output from zero to one (that is, when using a sigmoid whose output varies from zero to one), the desired output values are the original samples (with no noise added) scaled linearly from zero to one. This keeps the errors between zero and one. After training, the outputs are re-scaled to

the range of negative three to positive three to allow full reconstruction of the output signal.

A three layer multi-layer perceptron with 60 neurons in each layer is then trained to remove noise. After 200000 training iterations, training is stopped and the network tested. Testing data consists of generating new sine waves. In the first test, the three original sine waves are summed together and no noise is added to the resulting sine wave. In the second test, new data is generated, the three original sine waves are summed together, and noise is added. Finally in the third test, sine waves consisting of a single frequency with no noise are presented to the network.

Phase Two Testing. In phase two, the utterance of the word "one" is digitized (8 bits) at a 8 Khz rate for a total of 2280 discrete samples. The network scans the utterance taking 60 samples at a time, from the beginning to the end. When the end of the utterance is reached, the network returns to the beginning for another pass. One complete pass is considered to be one training iteration. As before, uniform pseudo random noise is added to achieve a signal-to-noise ratio of 0 db.

Again, a three layer multi-layer perceptron consisting of 60 neurons in each layer is trained to remove the noise. After 250000 training iterations (a total of 9.5 million presentations of 60 samples each are used), training is stopped and the original along with other speech samples are presented to the network, with and without noise to test the

performance of the multi-layer perceptron. Finally, as in the phase one testing the inputs and outputs are scaled as necessary.

Results and Discussion

Figure 38 shows the output from the phase one trained network at the original frequencies with no noise added. The output is very close to the original waveform. When noise is added to the signal, the output again appears to resemble the original waveform as seen in Figure 39.

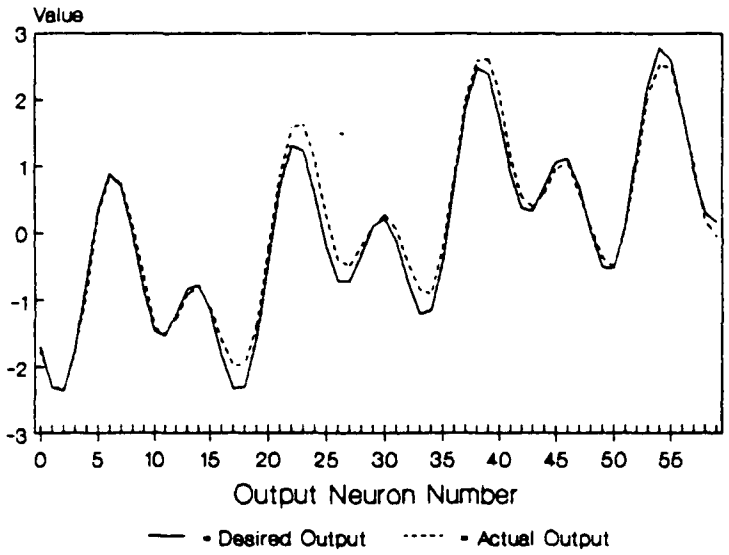


Figure 38. Output With No Noise Added.

If different frequencies are tried, the output no longer matches the original waveform. Figure 40 shows the output versus the input for a noise free signal generated by three sine waves at the frequencies 50 Hz, 750 Hz, and 1500 Hz. As seen in Figure 40, there is the presence of the three original frequencies. There are approximately 7.5 small cycles (caused by the 1000 Hz training), 4 larger cycles (caused by

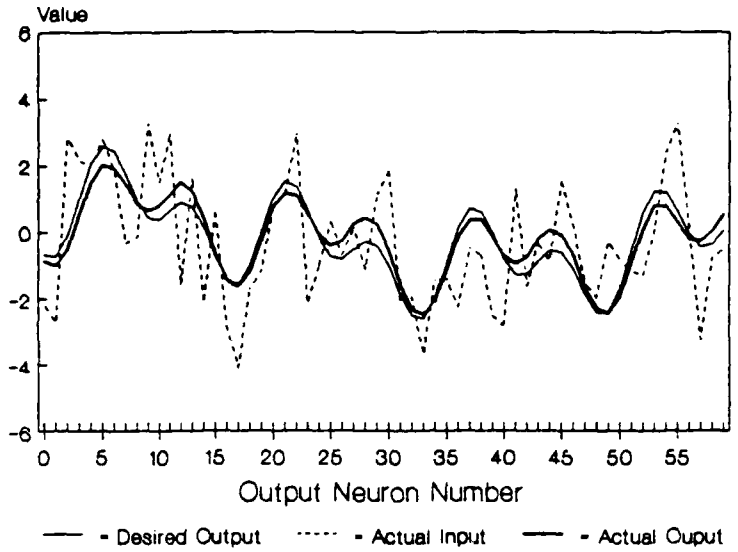


Figure 39. Output With Noise Added.

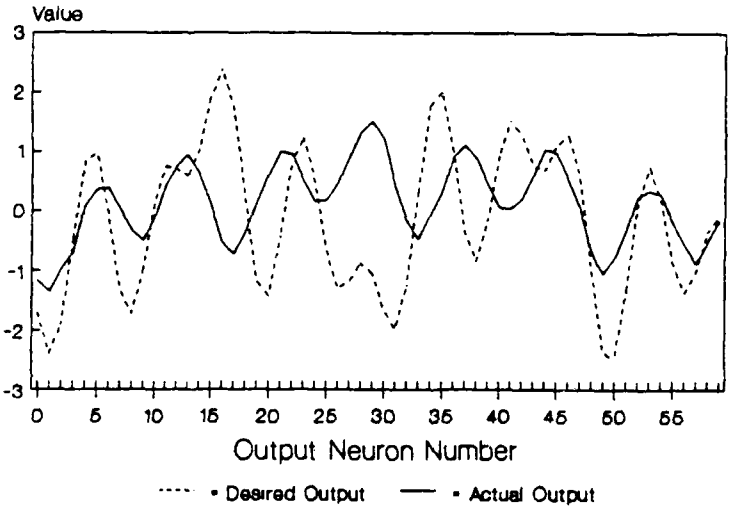


Figure 40. Output When the Input Frequencies are 320, 810, and 1330 Hz With No Noise Added.

the 500 Hz training), and a curving affect (caused by the 100 Hz training) for this input. Other sets of input frequencies show the same characteristic output as seen in Figure 40 (see Figures 41 and 42).

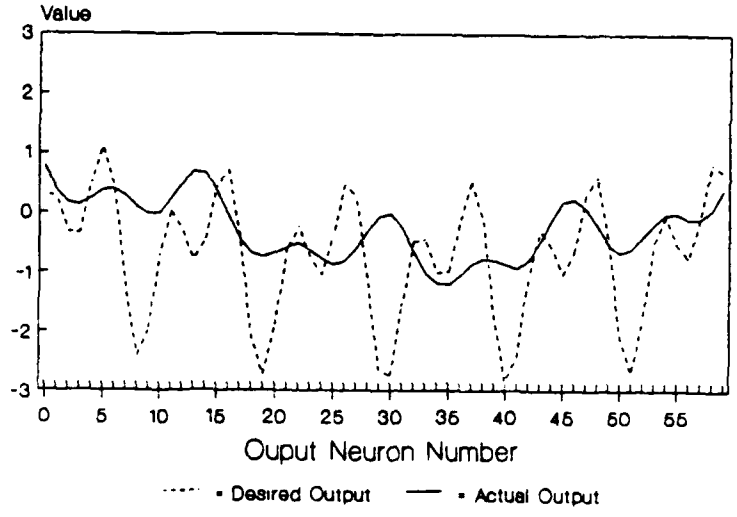


Figure 41. Output When the Input Frequencies are 50, 750, and 1500 Hz With No Noise Added.

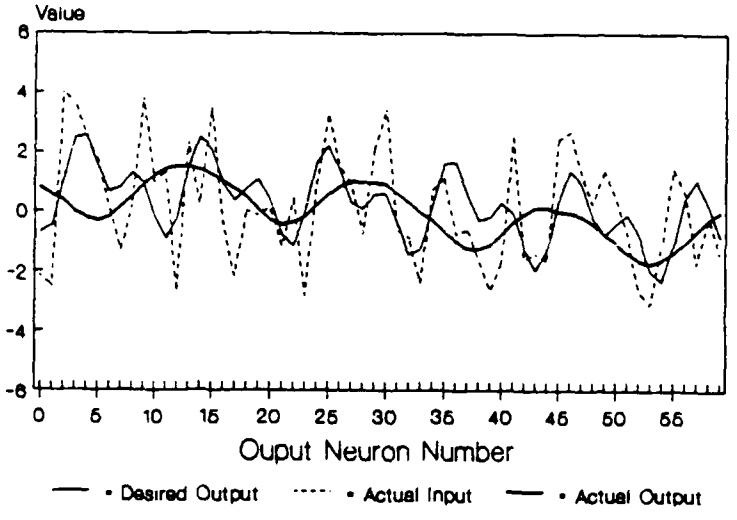


Figure 42. Output When the Input Frequencies are 50, 750, and 1500 Hz With Noise Added.

When individual sine waves are introduced, the effect becomes more pronounced. Figures 43 and 44 clearly demonstrate that the multi-layer perceptron has a "high Q" factor at each of the three original frequencies. It is also clear that the network is still filtering, but unfortunately the

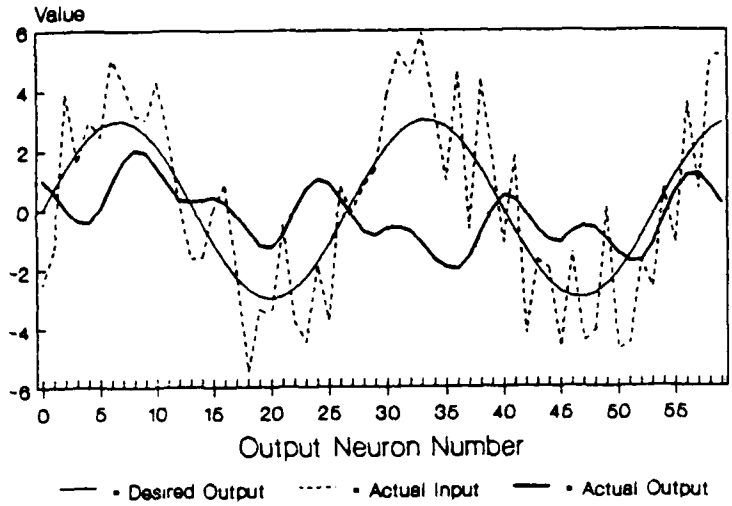


Figure 43. Output When the Input Frequency is 300 Hz With Noise Added.

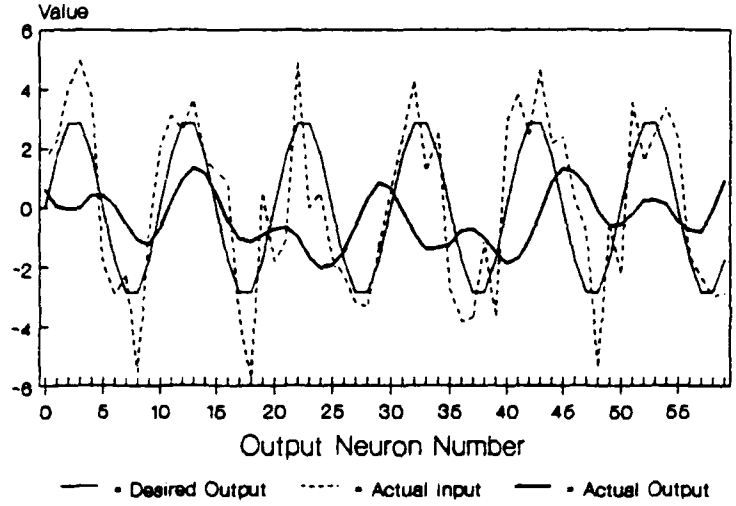


Figure 44. Output When the Input Frequency is 800 Hz With Noise Added.

output seems to consist primarily of a "ringing" at the three original frequencies.

Figures 45 and 46 show the reproduction of the speech samples from the phase two testing. As shown, the output signals do not match the input waveforms. Despite the

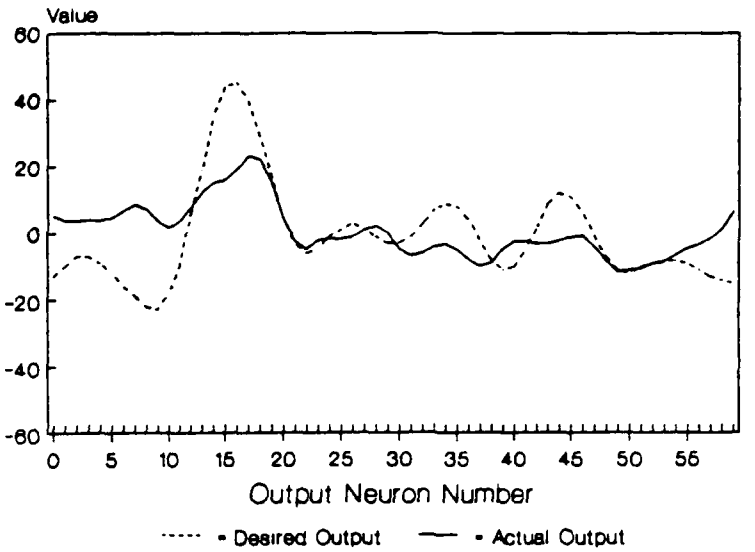


Figure 45. Part of the Word "One" With No Noise Added.

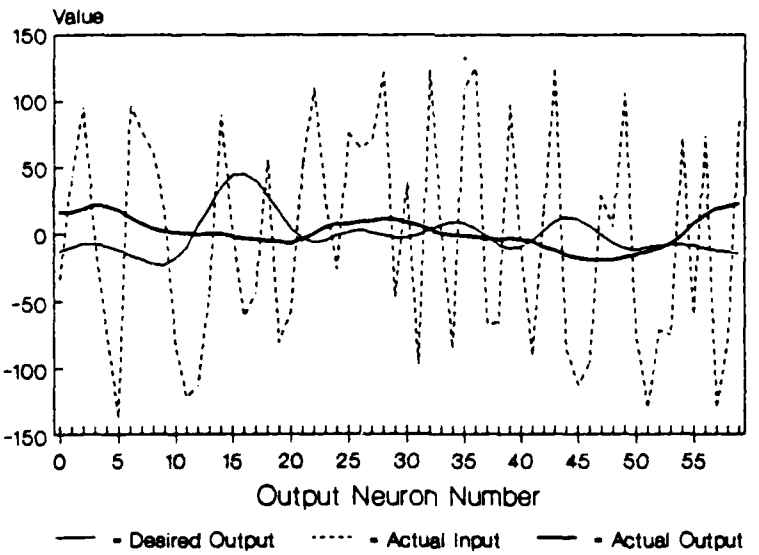


Figure 46. Part of the Word "One" With Noise Added.

lengthy training, the network is not able to handle all the various frequencies that are found in the speech waveform. This may be a function of the number of neurons used in the hidden layers. Therefore, different network sizes should be tested to determine whether the number of frequencies that can be resolved might be a function of the number of neurons

used.

In conclusion, it has been shown that the multi-layer perceptron will resonate at the trained frequencies and will produce signals at these frequencies regardless of the input. There may also be a limit to the number of frequencies that this type of network can accommodate. Also, since these experiments were unable to reproduce the results obtain by Tamura and Waibul, it is recommended that the method they describe not be used for noise reduction since other methods exist that require less computation and obtain better results.

Finally, the reader is also advised to see the thesis work of Captain Kevin Cox (1) for further descriptions of experiments in this topic area.

VI. Isolated Word Recognition Using a Multi-Layer Perceptron and a Kohonen Self-Organizing Feature Map

Introduction

The purpose of this chapter is to develop an isolated word recognition system using a multi-layer perceptron and a Kohonen self-organizing feature map. The first section briefly discusses the structure of the system and how it functions. The second section describes the training of the neurons for the system. The third section describes the testing that is to be performed. The last section presents the results and recommendations for future use.

Background

It has been noted that the spectrograms for the same English word spoken by independent speakers are very similar (6:45) and it was once believed that these spectrograms could be used by the deaf as a new language (7:312). Figure 47 shows the spectrogram for the phrase "We are here" spoken by independent speakers. If the spectrograms for the same word are similar, it is hypothesized that these Fourier representations of the word can be mapped to the same location on a Kohonen network since, as mentioned in chapter one, a Kohonen network has the property of grouping similar inputs together.

The system to be discussed in this chapter is based on

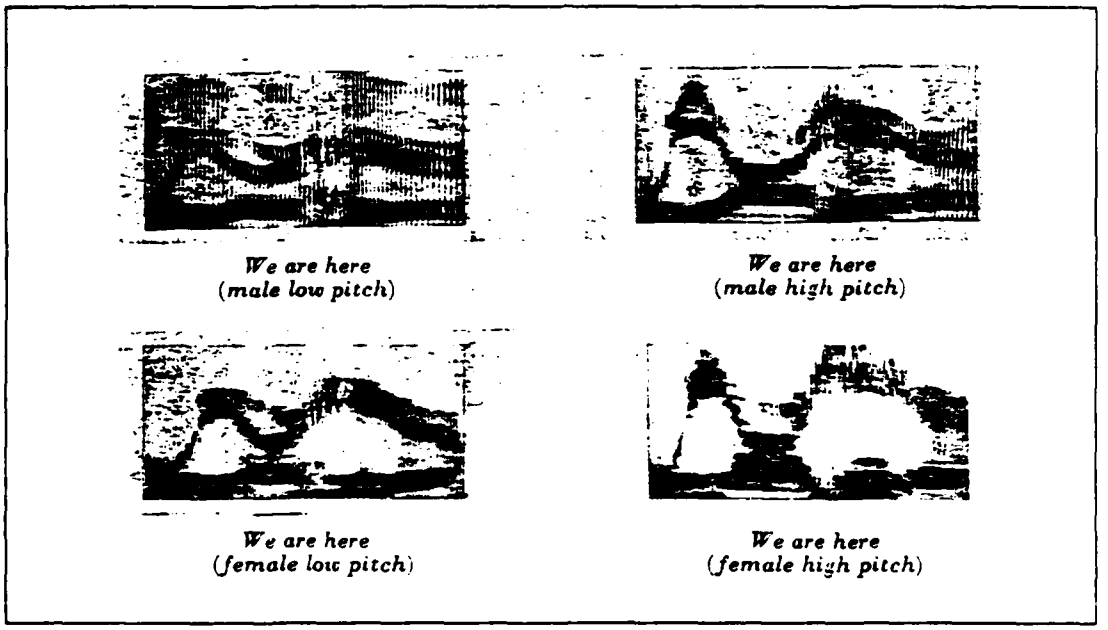


Figure 47. Spectrograms for the Phrase "We are here." (From 6:45)

using a 10-by-10 Kohonen network to store the spectrograms for the words "zero" through "nine" of one speaker. When the spectrogram of a word enters, it is stored temporarily in short-term memory. The spectrogram is then "seen" by each neuron in the Kohonen network. The weights of one of the neurons in the Kohonen network will be closer to an input spectrogram than the other neurons, when measured in Euclidean distance (or dot product). This neuron will then have the maximum output relative to the other neurons for the Kohonen network. By using two MAXNET's above the Kohonen network, the x and y coordinate of the winning neuron can be established. These coordinates are fed to a three layer multi-layer perceptron consisting of 40 neurons in the first layer, 40 neurons in second layer, and 10 neurons in the output layer. The multi-layer perceptron then

transforms the location from the Kohonen network to an output neuron. That output neuron's position is such that it represents the word that was spoken. Thus, the output neurons for the multi-layer perceptron are labelled "zero" through "nine". Basically, the multi-layer perceptron is acting as a reference table, converting location from the Kohonen network to an output which represents the numerical value of the word spoken. Figure 48 shows a diagram of the system.

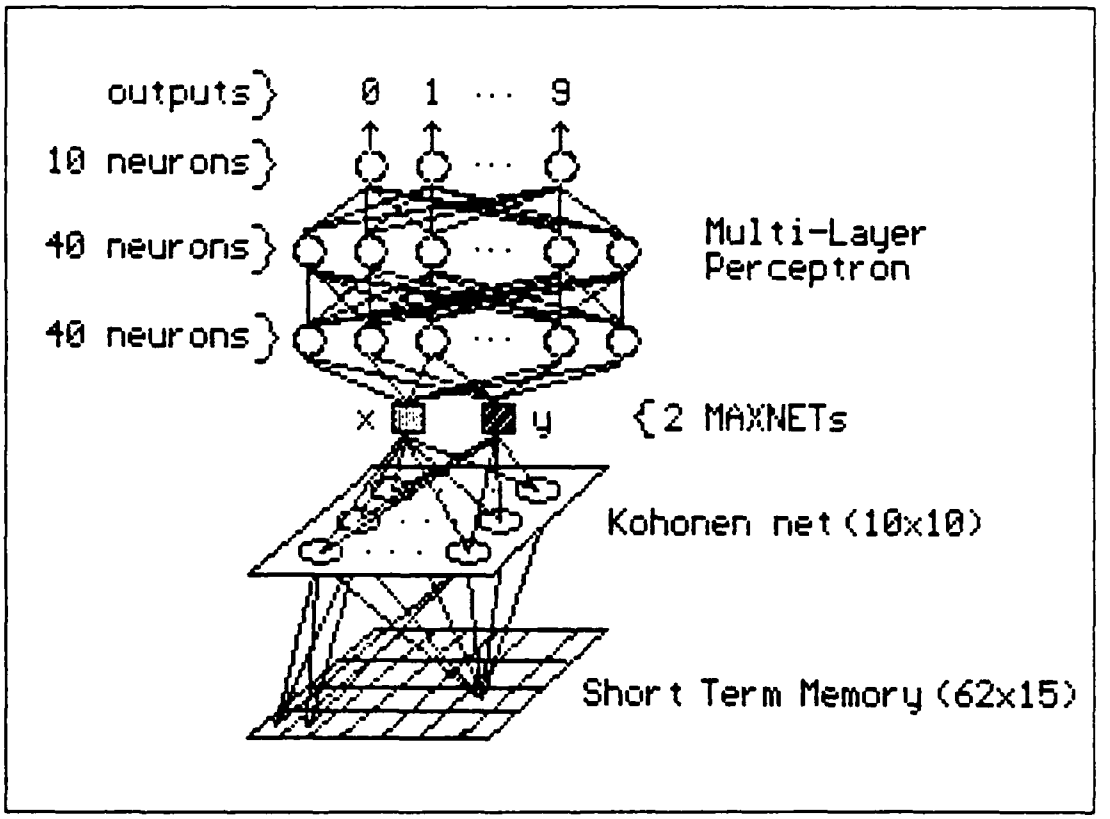


Figure 48. Diagram of the Speech Recognition System.

Before any training or testing, all speech files are filtered using a digital lowpass FIR filter that has a cutoff frequency of 3 KHz with the results being stored in

new files. In addition, where each word begins and ends in all the files is recorded beforehand while viewing the speech waveforms on a CRT.

Training

Training consists of selecting a word at random from a filtered file of ten digitized (8 bit) words sampled at 8 KHz. Next, two random numbers ranging from -500 to 500, are added one each to the start and stop locations of the randomly chosen word. This method of selecting a sample word enables segmenting routines to have some leeway in selecting the starting and stopping locations. This method also allows the Kohonen network to store numerous "views" of the spectrograms of the words at various lengths, instead of storing the same "view" for each of the ten words if the same starting and ending locations been chosen.

Next, the length (stop - start) of the utterance is obtained and is used to compute a ratio (either up or down sampling the utterance chosen) such that the total number of Fourier transforms (62 transforms of 64 samples each, representing 0.496 seconds of speech) is always the same. This has the effect of uniformly expanding or compressing each utterance to the same length, similar to the effect found in Dynamic Time Warping.

From this scaled utterance, 64 samples are taken and the Fast Fourier Transform is then computed. Next, the magnitudes of the positive frequency coefficients are computed.

These magnitudes are then logarithmically added together into 15 separate coefficients to compensate "...for the ear's decreasing frequency resolution with increasing frequency" (7:300). Finally, the resulting 15 coefficients are energy normalized. These coefficients are stored in an array and the next 64 samples are retrieved. The process is repeated until a total of 62 Fourier transforms are taken (which represents 0.496 seconds of speech). This produces an 62x15 array that represents the spectrogram for an utterance stored in short-term memory.

The 62x15 array is then fed to the Kohonen network. Since each neuron in the Kohonen network will store a spectrogram, each neuron will have 930 weights (62*15 = 930). Next, the Kohonen network goes through its training which consists of selecting the neuron whose weights are the closest to the input when measured in Euclidean distance. In addition, those neurons that are in the neighborhood of the winning neuron have their weights slightly adjusted such that they move closer in Euclidean distance to the winning neuron. The neighborhood size originally starts with one half of the size of the network. As time progresses, this neighborhood size decreases until only the winning neuron is updated. During the last 1000 training iterations, the word that was randomly chosen is recorded along with the winning neuron location from the Kohonen network. This provides a long-term memory map of word versus location by which the multi-layer perceptron can be trained, and is shown in Fig-

ure 49.

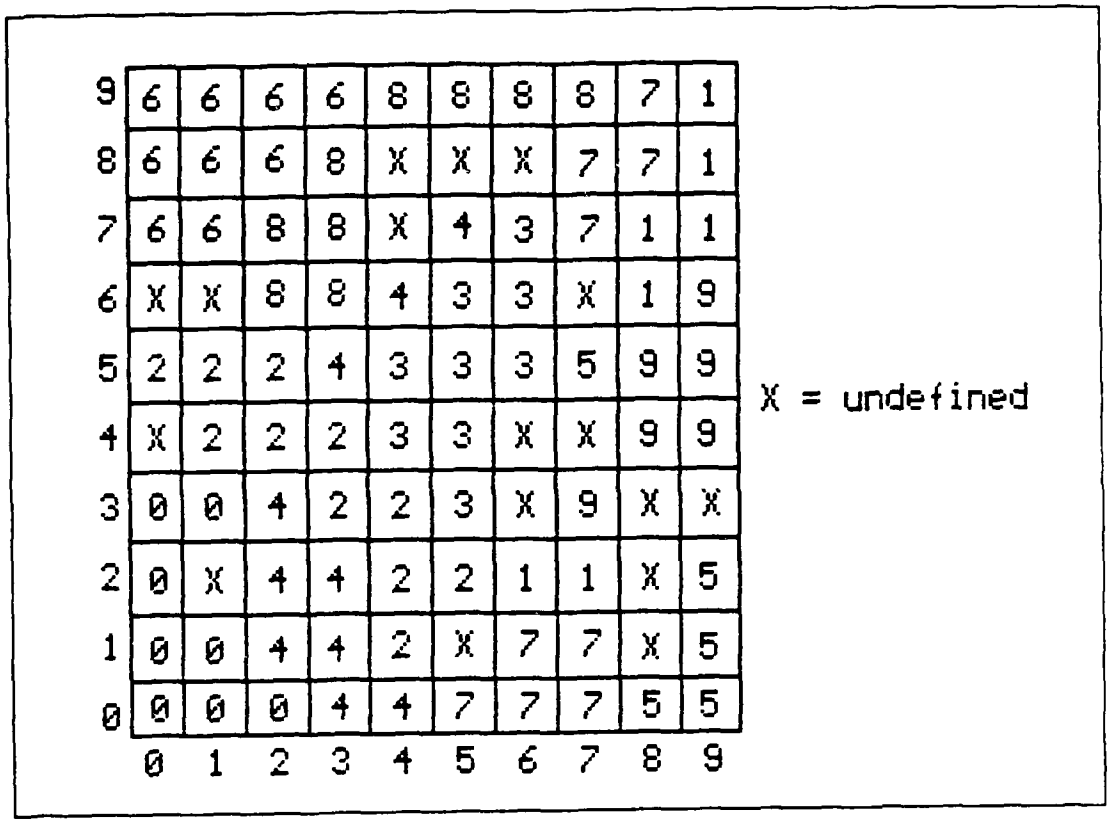


Figure 49. Memory Map.

Training for the multi-layer perceptron begins by randomly selecting a location from the map. Two small random values are generated ranging from -0.4 to 0.4. One of the random numbers is added to the x coordinate and the other to the y coordinate. This ensures that each recorded location from the map is surrounded by hyperplanes within the multi-layer perceptron, since errors may be occur if the hyperplanes are to close to a recorded location. The output from the multi-layer perceptron is checked to ensure that the word computed matches the word chosen from the map. If not, the multi-layer perceptron corrects the weights and thresh-

olds of its neurons. Note, if a position in the map has no word associated with it, a different location is chosen randomly.

Testing

During testing three different digitally filtered speech files are used. One file is from the original male speaker that was recorded at a different time than the one used in training. The other two 8 Khz files are from a female and a different male speaker. In each case, the starting and stopping location of each word is known.

Each speaker is tested in a similar manner: first, using the known starting and ending locations for each of the 10 words, then using locations that are within plus or minus 500 samples of the known starting and stopping locations for each word.

Results and Discussion

When the network is tested by using the original speaker at the known starting and ending locations for each word, the network achieves 100 percent correct word identification as shown in Table III. However, when different starting and ending locations are used, the accuracy starts to fall off as seen in Table IV. In one case, as shown in Table V, only the ending location for the word "zero" is changed and the wrong answer is produced; whereas in the previous sample, a lower starting location (see Table IV) is used and the cor-

rect answer is achieved. The overall accuracy for five tests is 80 per cent.

Table III. First Male Speaker Using Known Starting and Ending Locations

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	940 - 4420	0	0,1
1	4780 - 8260	1	9,9
2	8900 - 11860	2	3,4
3	13180 - 16200	3	5,4
4	17640 - 20480	4	2,2
5	21800 - 24420	5	9,2
6	25480 - 28900	6	0,8
7	30120 - 33200	7	6,0
8	35050 - 37480	8	4,7
9	38940 - 42160	9	9,5

Table IV. First Male Speaker Using Different Starting and Ending Locations - Set One

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	900 - 4000	0	1,2
1	4500 - 8500	1	9,8
2	8600 - 11560	2	2,4
3	13000 - 16000	3	4,4
4	17400 - 20680	4	2,2
5	21300 - 24900	9	9,3
6	25000 - 29100	6	0,8
7	30600 - 33000	7	7,7
8	35300 - 37200	8	7,9
9	38540 - 41760	9	8,4

Table V. First Male Speaker Using Different Starting and Ending Locations - Set Two

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	940 - 4000	4	2,1
1	4980 - 8100	1	9,9
2	9200 - 12160	3	4,5
3	13400 - 16400	3	6,5
4	17840 - 20280	4	2,1
5	22000 - 24220	5	9,0
6	25700 - 28600	6	0,8
7	29900 - 33400	7	6,0
8	34850 - 37680	8	4,7
9	39340 - 42560	9	8,5

When the female speaker is tested first at the known starting and ending locations, as shown in Table VI, the accuracy drops off dramatically. Table VII shows a sample of the results when other locations are tried. Overall, out of five tests the accuracy is about 32 per cent.

Table VI. Female Speaker Using Known Starting and Ending Locations

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	3500 - 7340	5	8,2
1	7960 - 11160	9	9,4
2	12660 - 15420	7	6,0
3	17340 - 20100	3	5,4
4	22020 - 24760	5	8,0
5	26460 - 29200	9	9,3
6	30940 - 33980	6	0,8
7	35420 - 38700	7	6,1
8	40140 - 42480	8	3,7
9	44000 - 46740	9	9,4

Table VII. Female Speaker Using Different Starting and Ending Locations

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	3600 - 7280	5	8,2
1	8290 - 11500	5	7,5
2	13000 - 15750	3	4,5
3	16900 - 20480	7	5,1
4	22300 - 24950	2	8,8
5	26860 - 28800	5	9,1
6	30640 - 33800	6	0,7
7	35000 - 39100	2	0,5
8	40400 - 42280	8	6,9
9	43800 - 46540	9	8,4

Finally, the other male speaker is tested using known starting and ending locations; the accuracy of the network again drops, as seen in Table VIII. Table IX shows a sample of the results when different locations are tried. Overall, the accuracy of the network is about 44 per cent out of five tests.

Table VIII. Second Male Speaker Using Known Starting and Ending Locations

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	2840 - 5980	7	5,1
1	7060 - 10080	5	9,0
2	11900 - 14480	7	7,8
3	16500 - 19360	7	7,8
4	21380 - 23920	4	3,1
5	25680 - 28720	5	9,1
6	29520 - 33480	6	1,6
7	34120 - 37720	7	7,0
8	39040 - 41060	7	7,8
9	42900 - 46120	9	9,6

Table IX. Second Male Speaker Using Different Starting and Ending Locations

Desired Output	Location Entered	Actual Output	Kohonen Neuron Selected
0	2900 - 5600	0	1,3
1	7100 - 9900	1	9,7
2	11800 - 14800	2	2,5
3	16600 - 19500	7	7,8
4	21600 - 24000	4	4,0
5	25180 - 29220	5	9,2
6	29600 - 33700	6	1,6
7	34000 - 38000	5	8,2
8	38900 - 41260	7	7,8
9	42400 - 46620	9	8,3

The system appears to be fairly accurate at speaker dependent word recognition. One benefit of this system is that it allows whole word recognition to be performed rather than having to find good phoneme samples as in other speech recognition systems.

Improvements can be made in this system, since other tests have shown that using the Euclidean distance of the spectrograms is not a good measurement to use for word matching whether for individual or different speakers. It is recommended that the neurons in the Kohonen network use more than just Euclidean distance. This may help alleviate the problem of matching words among different speakers.

Also, as seen in Figure 49, the words could be distributed throughout the map. Therefore, it is recommended that the Kohonen map be divided into equal areas for each word before training is started. Then as training proceeds, force (that is, supervised the training) the Kohonen network to map words into their preselected areas. This will group

the words together and make the problem of transforming location to word easier, while creating generic templates for each word. "Supervised retuning" of the Kohonen network for additional speakers may also alleviate the problem of speaker dependency.

Finally, a single-layer perceptron should suffice to translate location to word output instead of requiring a multi-layer perceptron.

VII. Improvement of the Basic Neuron

Introduction

The purpose of this chapter is to discuss how the basic neuron model found in the single-layer and multi-layer perceptron can be improved to increase its computational power. The first section will introduce the alterations made in the basic neuron model. The second section describes the changes in the training rules for the single-layer and multi-layer configurations. The third section will introduce some added capabilities that are provided by the new neuron model. The fourth section describes the testing to be performed. The last section presents the results of testing and the need for further research.

Improvements

It is interesting to note that the equation shown in the article by Lippmann (5:13) to calculate the actual output of the perceptron uses one weight for every input. Looking closer at this equation reveals the weights are multiplying functions of the inputs. That is, x_0 can be expressed as $x_0 + 0 * x_1 + 0 * x_2 + \ldots + 0 * x_n$, and so on for each input. The question that arises is: "Can any function composed of the inputs be tied to these weights?" If so, then perhaps the neuron could increase its computational power.

Therefore, the major change proposed here for the basic neuron is the introduction of function generators, of which

any number can be used, as seen in Figure 50. These function generators shared the inputs and use any combination of

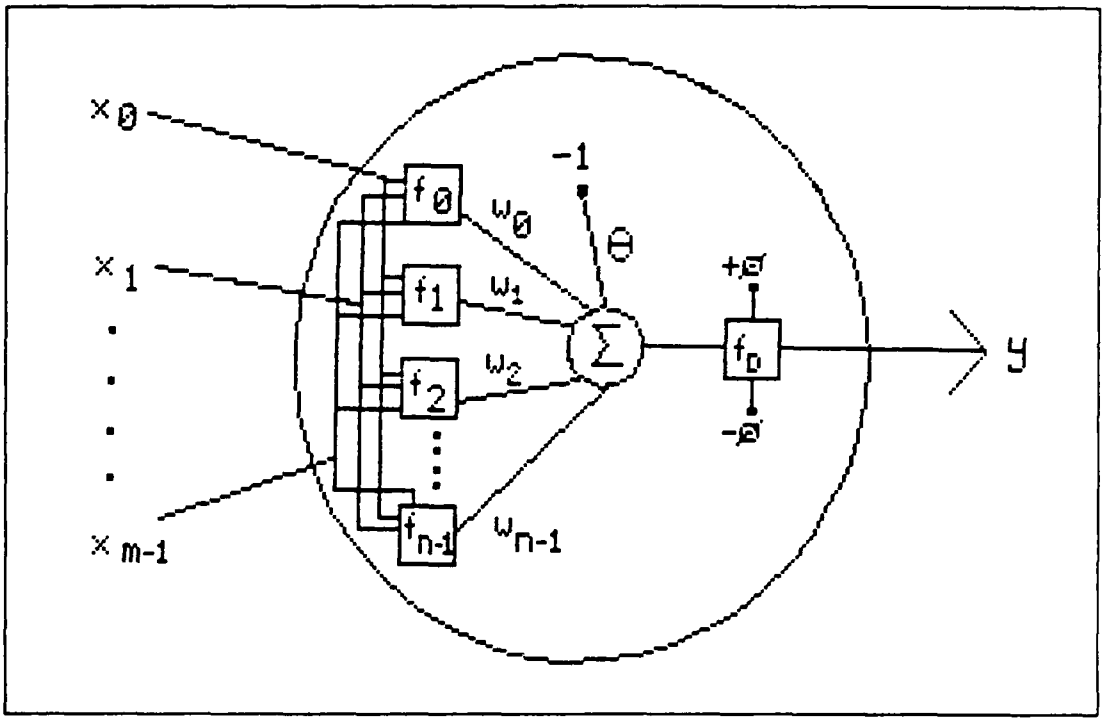


Figure 50. Usual Neuron With Changes.

the inputs as needed. For example, some of the functions that may be provided are:

$$f_0(x_0,x_1,\ldots,x_{m-1}) = x_0, \quad (13)$$

$$f_1(x_0,x_1,\ldots,x_{m-1}) = x_1 * x_2, \quad (14)$$

$$f_2(x_0,x_1,\ldots,x_{m-1}) = \sin(x_0), \quad (15)$$

$$f_3(x_0,x_1,\ldots,x_{m-1}) = x_1, \quad (16)$$

$$f_4(x_0,x_1,\ldots,x_{m-1}) = \text{random}(), \quad (17)$$

and so on. To convert the new neuron model, referred to subsequently as a "neurolog", to a basic neuron all that is required is to have the weights of the function generators, except those that only use the inputs, be set to zero. Random numbers could also be used, as in equation (17). A

random number generator may be helpful for when a neurolog is required to generate its own output with little or no inputs.

The output of the function generators are then multiplied by the weights attached to these generators and summed together with a threshold. The summation output is then fed to a discriminative function and the output is created.

As training progresses, all of the function generators provide outputs in parallel. The neurolog then selects those functions that will help solve the problem at hand. Those functions contributing will have their weights updated, while all other unnecessary functions will have their weights set to approximately zero.

The discriminative function may be either the hard limiter, as seen in Figure 51, or the sigmoid function, as seen in Figure 52.

There is an added feature that is has not been exploited with the usual neuron, namely the possibility of a multi-level output. A multi-level output would enable a single neurolog to discriminate more than one class. This capability is the consequence of the more intricate discriminative functions made possible by non-planer hypersurfaces; this will be discussed later in more detail.

Finally, one neurolog can be combined with other neurologs so that even more complex hypersurfaces can be created through the computational power found in a multi-layer perceptron.

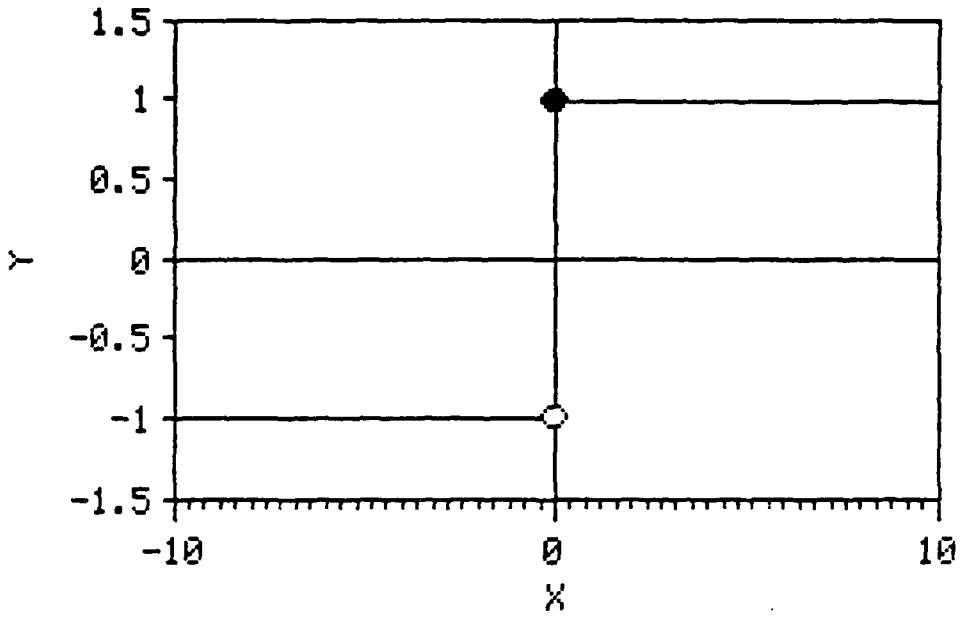


Figure 51. Hard Limiter Function.

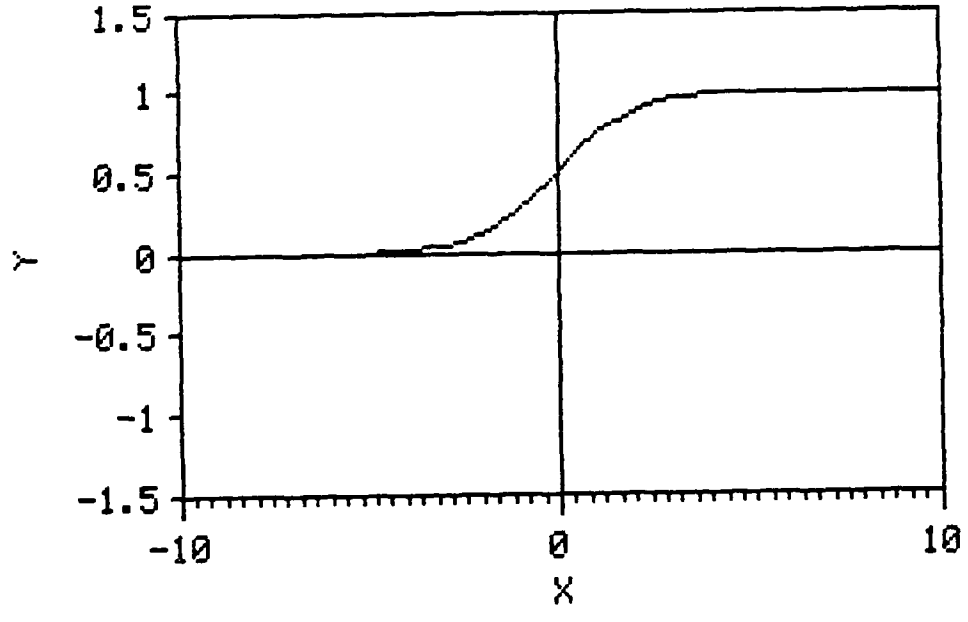


Figure 52. Sigmoid Function.

Training

The following is based on the work of Rosenblatt, Werbos, Parker, Rummelhart, and others who developed the current training rules used by the single and multi-layer perceptrons.

Initially, the constant ϕ in Figure 50 will be assumed to be zero and therefore, only a two level output is considered.

First, the function generators may use any combination of inputs, such as those equations mentioned in (13) to (17), that the user believes may help in solving the problem. Caution is required since some functions will not accept all values; for instance, log x_i would require some apriori knowledge that the range of x_i is greater than zero.

Next, set the weights and thresholds to small random values (typically between -0.5 and +0.5). Then present the new input and the new desired output. Finally, calculate the output using either the hard limiter or the sigmoid function. That is,

for the hard limiter

$$y(t) = f_h\left(\sum_{i=0}^{N} w_i(t)\ f_i(x_0(t),x_1(t),\ldots,x_{m-1}(t)) - \theta\right) \quad (18)$$

for the sigmoid

$$y(t) = f_s\left(\sum_{i=0}^{N} w_i(t)\ f_i(x_0(t),x_1(t),\ldots,x_{m-1}(t)) - \theta\right) \quad (19)$$

where N is the number of function generators, $w_i(t)$ is the weight i tied to the function generator at time t, m is the number of inputs which may differ in number with the number of weights, θ is the threshold, f_h is the hard limiter

function, and f_s is the sigmoid function.

To adapt the weights if using the hard limiter:

$$w_i(t+1) = w_i(t) + \eta\,[d(t) - y(t)]\, f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) \qquad (20)$$

where η is the gain (usually 0.2 to 0.4 and must be positive) and d(t) is the desired output (typically -1 or +1, though 0 and +1 can be used) at time t.

To adapt weights if using the sigmoid function (this also applies to the output nodes in a multi-layer perceptron configuration) :

$$w_i(t+1) = w_i(t) + \eta\,[d(t) - y_i(t)]\,[y_i(t)]\,[1-y_i(t)]\, f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) \qquad (21)$$

where all the terms are those as previously defined. Note, it may be possible to use a momentum term, though this has yet not been tested.

If using a multi-layer perceptron configuration, the hidden layers use:

$$w_j(t+1) = w_j(t) + \eta\,[y_j(t)]\,[1 - y_j(t)]\,[\Sigma_k \delta_k (\Sigma_q w_q)]\, f_j(x_0(t), x_1(t), \ldots, x_{m-1}(t)) \qquad (22)$$

where $y_j(t)$ is the output of the hidden node j at time t, δ_k is the error at node k in the layer above, Σw_q is the sum of the weights in node k in the layer above that use (in the function generator of node k) the output of node j, and $f_j(x_0(t), x_1(t), \ldots, x_{m-1}(t))$ is the output of the function generator connected to weight j that is being updated. Again, the momentum term has not been tried.

Thresholds likewise can be updated if desired. In any

case (this applies for the sigmoid case, the hard limiter case, and for the thresholds in the output layer of a multi-layer perceptron) the thresholds are updated by:

$$\theta(t+1) = \theta(t) - \eta\,[d(t) - y(t)] \qquad (23)$$

For the thresholds in the hidden layers of a multi-layer perceptron use:

$$\theta_j(t+1) = \theta_j(t) - \eta\, y_j(t)\,[1 - y_j(t)]\,[\sum_k \delta_k (\sum_q w_q)] \qquad (24)$$

where $\theta_j(t)$ is the threshold of the hidden node j at time t.

Other Capabilities

As mentioned before, the sigmoid function has the form:

$$y = \epsilon \,/\, (1 + \exp-(\beta\alpha + \phi)) - \mu \qquad (25)$$

where $\alpha = w_i f_i(x_0, x_1, \ldots, x_{m-1}) - \theta$, ϵ is the amplitude of the sigmoid, β is the rate at which the sigmoid rises, ϕ will allow the sigmoid to shift left or right on the α axis, and μ will move the sigmoid up or down on the y axis. Typically, ϵ is one, β is one, ϕ is zero, and μ is zero.

There are several possibilities that exist using the general form of a sigmoid. First, β can be increased (or decreased) or β can be allowed to adapt. Likewise, ϕ can be set or also allow to adapt.

β training is optional and the user has the choice to either set the β term to another value greater than one or allow the neurolog to adapt the β term. Increasing β can be useful when the data regions are close to each other.

ϕ training is also optional and is used to generate multi-level outputs. It also possible to do ϕ training in

both the sigmoid and hard limiter case and this will be discussed later.

β Training

If an increase in β is to be used, it has been found useful to change the update rules for the weights to (used by both the single neurolog or the output layer in a multi-layer perceptron):

$$w_i(t+1) = w_i(t) - [\eta\beta/\epsilon]\,[y_i(t) + \phi]\,[y_i(t) + \phi - \epsilon]\,[d(t) - y_i(t)]\,f_i(x_0(t), x_1(t) \ldots, x_{m-1}(t)) \quad (26)$$

and for the thresholds

$$\theta_i(t+1) = \theta_i(t) + [\eta\beta/\epsilon]\,[y_i(t) + \phi]\,[y_i(t) + \phi - \epsilon]\,[d(t) - y_i(t)] \quad (27)$$

where all the terms are those previously mentioned. Likewise, for the hidden layers:

$$w_i(t+1) = w_i(t) - [\eta\beta/\epsilon]\,[y_i(t) + \phi]\,[y_i(t) + \phi - \epsilon]\,[\textstyle\sum_k \delta_k (\sum_q w_q)]\,f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) \quad (28)$$

and for the thresholds in the hidden layers:

$$\theta_i(t+1) = \theta_i(t) + [\eta\beta/\epsilon]\,[y_i(t) + \phi]\,[y_i(t) + \phi - \epsilon]\,[\textstyle\sum_k \delta_k (\sum_q w_q)] \quad (29)$$

again all terms are those previously mentioned.

Finally, if β training for each neurolog is desired then use:

$$\beta(t+1) = \beta(t) - \eta \{ [-1/(2\beta^2)]\,[d(t) - y(t)]^2 + [\alpha/\epsilon\beta]\,[d(t) - y(t)]\,[y(t) + \phi]\,[y(t) + \phi - \epsilon] \} \quad (30)$$

where $\alpha = w_i f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) - \theta(t)$ and all other

terms are those previously mentioned. For the hidden layers use:

$$\beta(t+1) = \beta(t) - (\eta\alpha/\epsilon)\ [y(t) + \phi]\ [y(t) + \phi - \epsilon]\ [\sum_k \delta_k (\sum_q w_q)] \qquad (31)$$

where all terms are those previously mentioned.

ϕ Training

Typically, ϕ is set to zero. However, by using a ϕ term and by allowing more than one sigmoid function, the neurolog can generate multi-level outputs. In turn, multi-level outputs allow for more than one class to be discriminated. The case considered here is to illustrate that multi-class discrimination is possible for a single neurolog. Therefore, the equations introduced do not necessary apply for a multi-layer perceptron configuration.

As shown below in Figure 53, that if two typical sigmoids are added together that three distinct levels are produced.

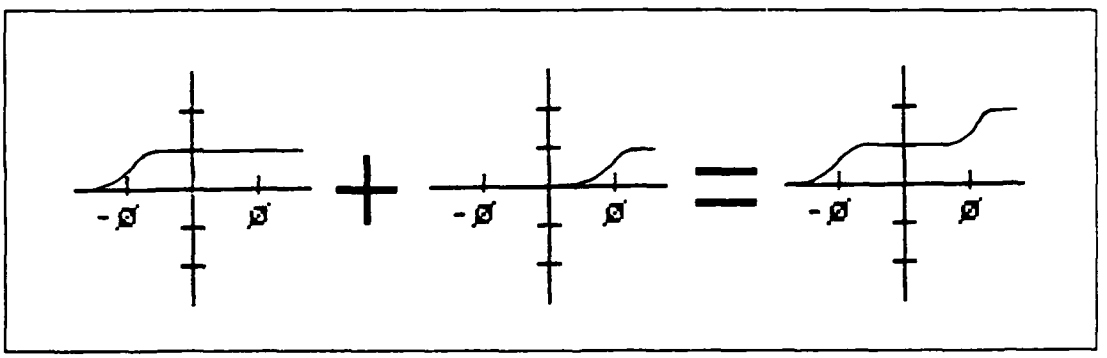


Figure 53. The Addition Of Two Sigmoids.

Thus, the summed sigmoids can be expressed as:

$$f(\alpha) = 1 / \{1 + \exp(-\alpha+\phi)\} + 1 / \{1 + \exp(-\alpha-\phi)\} \qquad (32)$$

As a result, the update rules become more complex. The update rule for the weights become:

$$w_i(t+1) = w_i(t) + \eta\,[d(t) - y(t)]\,[\exp(-\alpha_1) / (1 + \exp(-\alpha_1))^2 + \exp(-\alpha_2) / (1 + \exp(-\alpha_2))^2]\, f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) \quad (33)$$

where $\alpha_1 = [w_i(t) f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) - \theta] - \phi$,

and $\alpha_2 = [w_i(t) f_i(x_0(t), x_1(t), \ldots, x_{m-1}(t)) - \theta] + \phi$.

Likewise for the thresholds:

$$\theta(t+1) = \theta(t) - \eta\,[d(t) - y(t)]\,[\exp(-\alpha_1) / (1 + \exp(-\alpha_1))^2 + \exp(-\alpha_2) / (1 + \exp(-\alpha_2))^2] \quad (34)$$

where α_1 and α_2 are as previously mentioned. Finally, to allow ϕ to train use:

$$\phi(t+1) = \phi(t) + \eta\,[d(t) - y(t)]\,[-\exp(\alpha_1) / (1 + \exp(-\alpha_1))^2 + \exp(-\alpha_2) / (1 + \exp(-\alpha_2))^2] \quad (35)$$

The hard limiter case proceeds in the same fashion. If two hard limiters are added together, the result is a three level hard limiter function as seen in Figure 54. The new hard limiter can be expressed as:

$$f(\alpha) = (\alpha + \phi) / \text{abs}(\alpha + \phi) + (\alpha - \phi) / \text{abs}(\alpha - \phi) \quad (36)$$

where abs() is the absolute value and when α equals ϕ or $-\phi$, the resulting 0/0 equals 1.

In the hard limiter case, the weights and thresholds change as before (see equations (20) and (23)). The ϕ

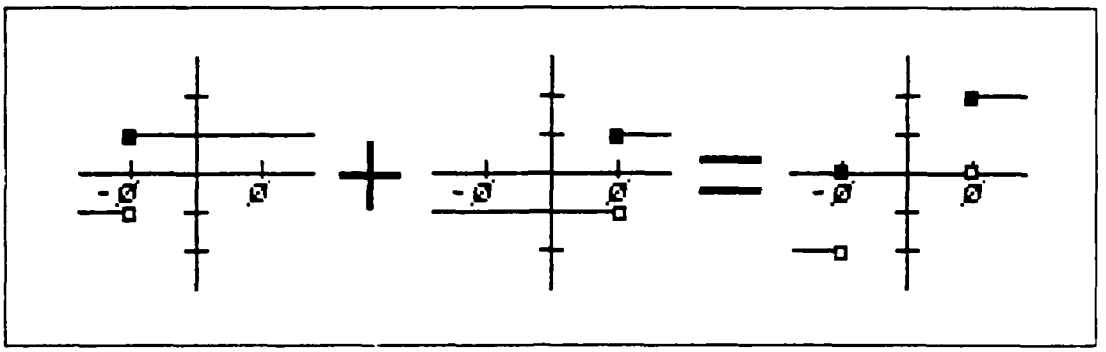


Figure 54. The Addition Of Two Hard Limiters.

training for the hard limiter case is:

$$\phi(t+1) = \phi(t) + \eta\,[d(t) - y(t)] \qquad (37)$$

Testing

First, a single neurolog is tested using a hard limiter to do the classical exclusive-or problem as seen in Figure 55. In this example, the function generators chosen are:

$f_0(x_0,x_1) = x_0$,

$f_1(x_0,x_1) = x_1$,

$f_2(x_0,x_1) = \sin(x_0)$,

$f_3(x_0,x_1) = \sin(x_1)$, and

$f_4(x_0,x_1) = x_0 * x_1$.

The weights (w_0,w_1,w_2,w_3,w_4) are all randomly set to values between -0.5 and +0.5. Since this is a two class problem, ϕ is not needed. After every 20 training iterations, training is stopped and 50 tests are performed to determine the neurolog's accuracy. Test data is chosen from one of the four data regions at random. Figure 56 shows the neurolog's accuracy as training progresses.

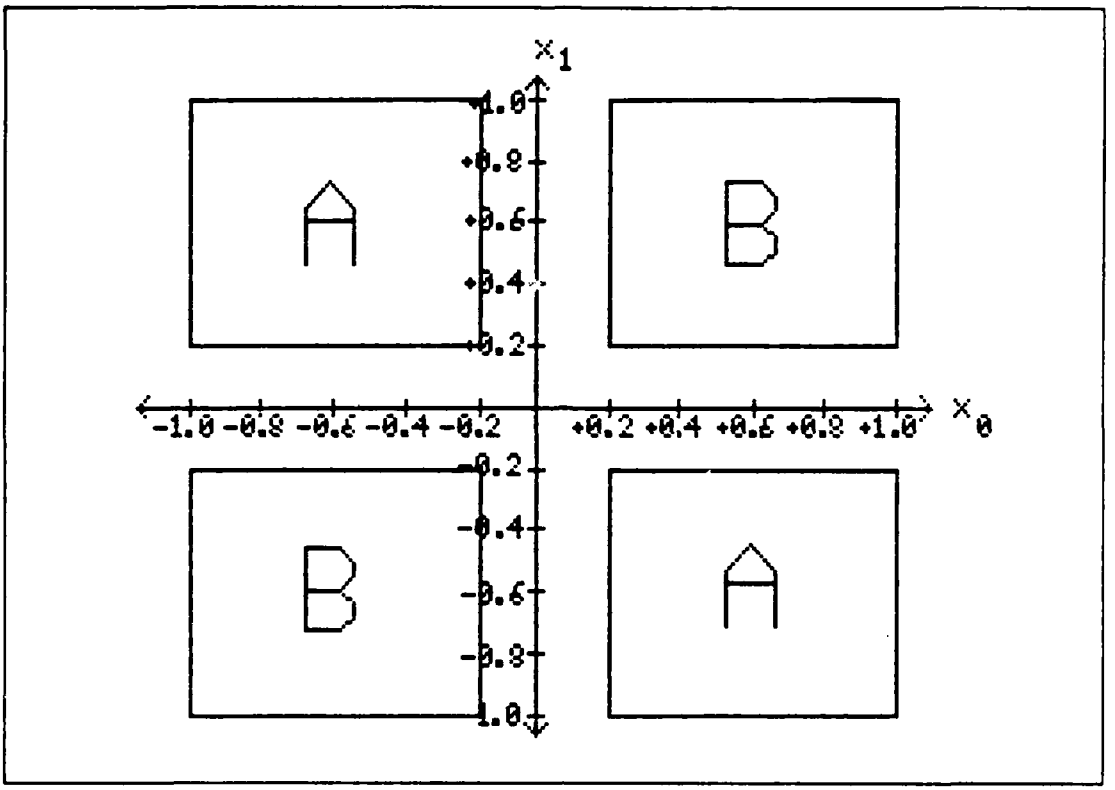


Figure 55. Exclusive-Or Data.

As can be seen in Figure 56, accuracy develops at an extremely high rate. This type of problem would take at least a two-layer perceptron to solve (training time would depend on the size of the net used). However, it can be seen that one neurolog can handle this problem. The resulting weights and thresholds (after 5000 training iterations) are:

w_0 = -0.265606,

w_1 = 0.071703,

w_2 = -0.149933,

w_3 = 0.216150,

w_4 = 3.587245, and

θ = 0.136775.

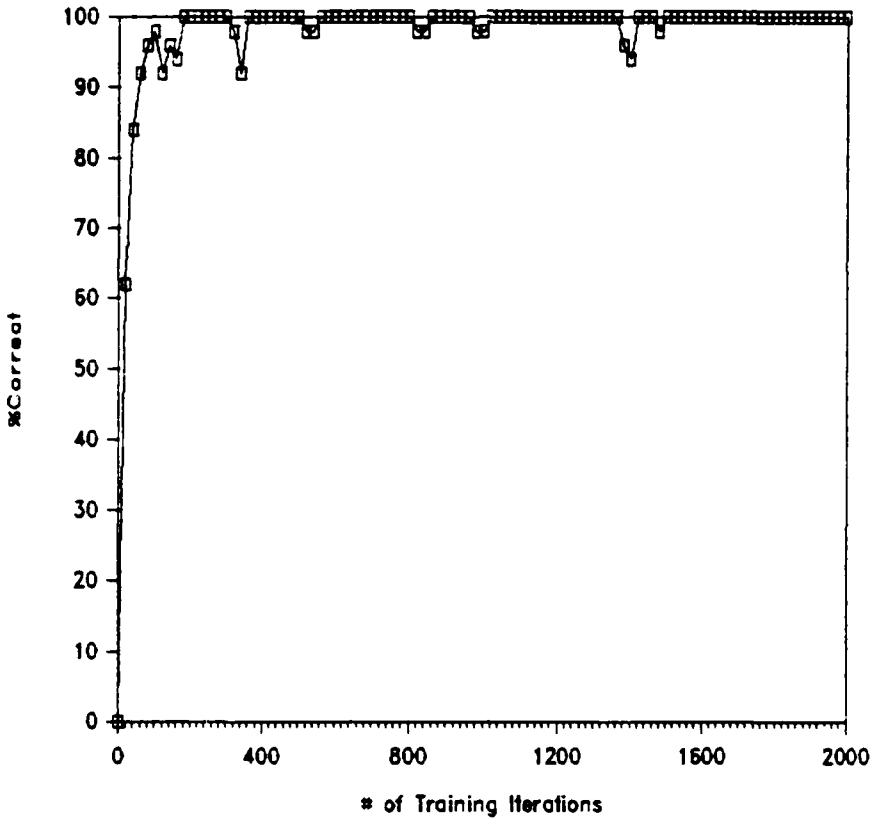


Figure 56. Training Accuracy for the Exclusive-Or Data Problem.

As might be expected the sine functions are not needed and the training procedure sets their weights to be small; that is also the case for the x_0 and x_1 function weights. By inspection of Figure 56, it can be seen that $x_0 * x_1$ can be used to tell the difference between these two classes and the resulting w_4 shows that the neurolog learned this fact.

The next example uses a three class problem as shown in Figure 57. This time the function generators used are:

$f_0(x_0,x_1) = x_0$,

$f_1(x_0,x_1) = x_1$,

$f_2(x_0,x_1) = x_0^2$,

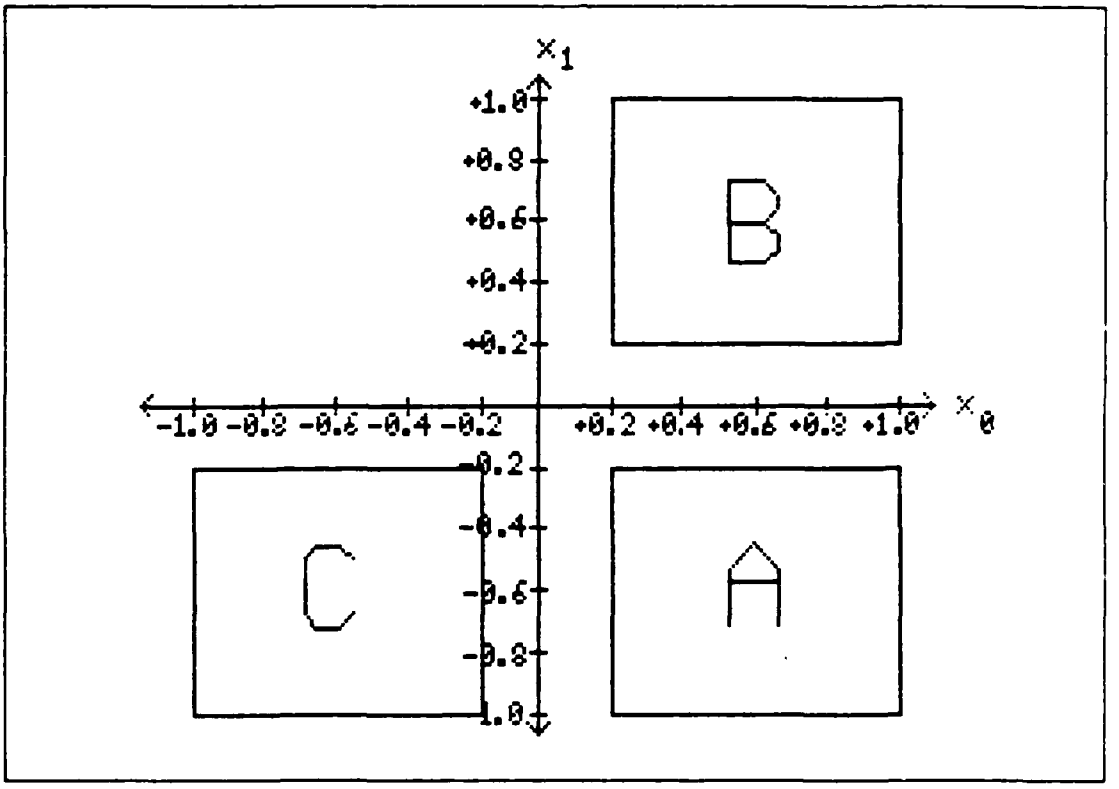


Figure 57. Three Class Problem.

$f_3(x_0,x_1) = x_1^2$, and

$f_4(x_0,x_1) = x_0 * x_1$.

The weights and threshold are set as before. Since the single neurolog is required to discriminate between three classes, ϕ is required and is randomly set between 1.0 and 2.0 (testing has shown that is best to start with ϕ greater than or equal to 1.0).

After every 10 training iterations, training is stopped and the neurolog is tested for accuracy. Figure 58 shows the neurolog's accuracy as training progresses. Though this problem is simplistic, it does serve to illustrate the capability that a single neurolog can provide. Figure 59 shows the neurolog's training accuracy when the desired outputs

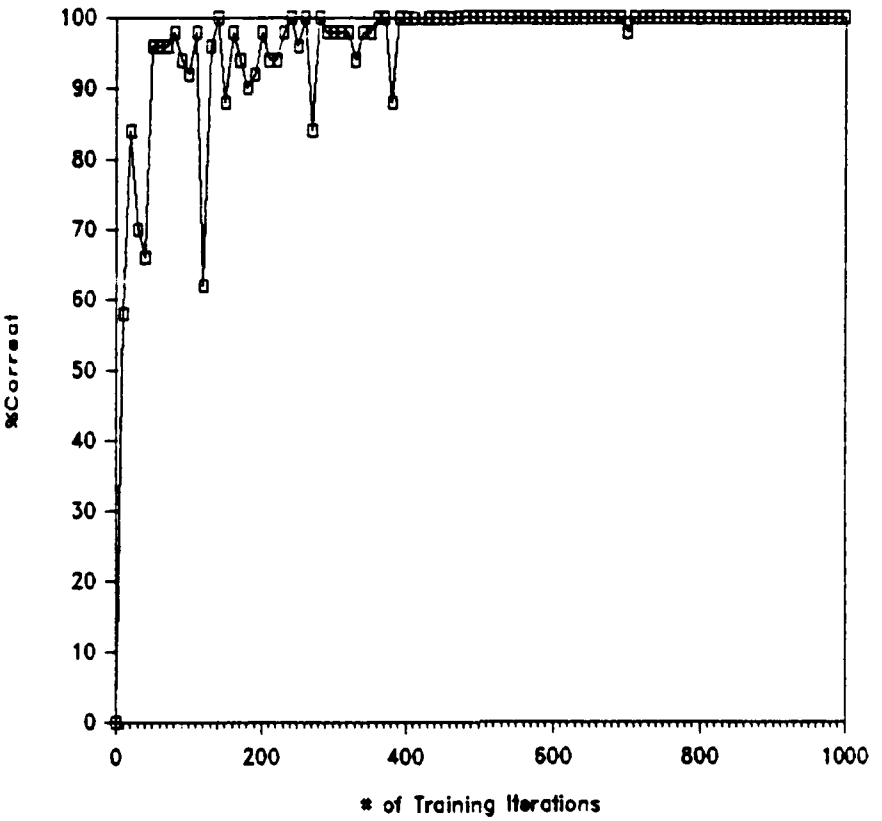


Figure 58. Training Accuracy for the Three Class Problem.

are "switched" such that Class A is +2, Class B is 0, and Class C is -2.0. This requires a much complicated complex hypersurface and therefore more training is required. Perhaps more functions could be used to increase accuracy and decrease training time since it is conceivable that a hypersurface does exist that would intrinsically separate the three classes.

Figure 60 shows the accuracy when using a sigmoid instead (rate (β) equals 5.0) of the hard limiter for the example shown in Figure 55. Figure 61 shows the accuracy when using a neurolog (also for the example shown in Figure

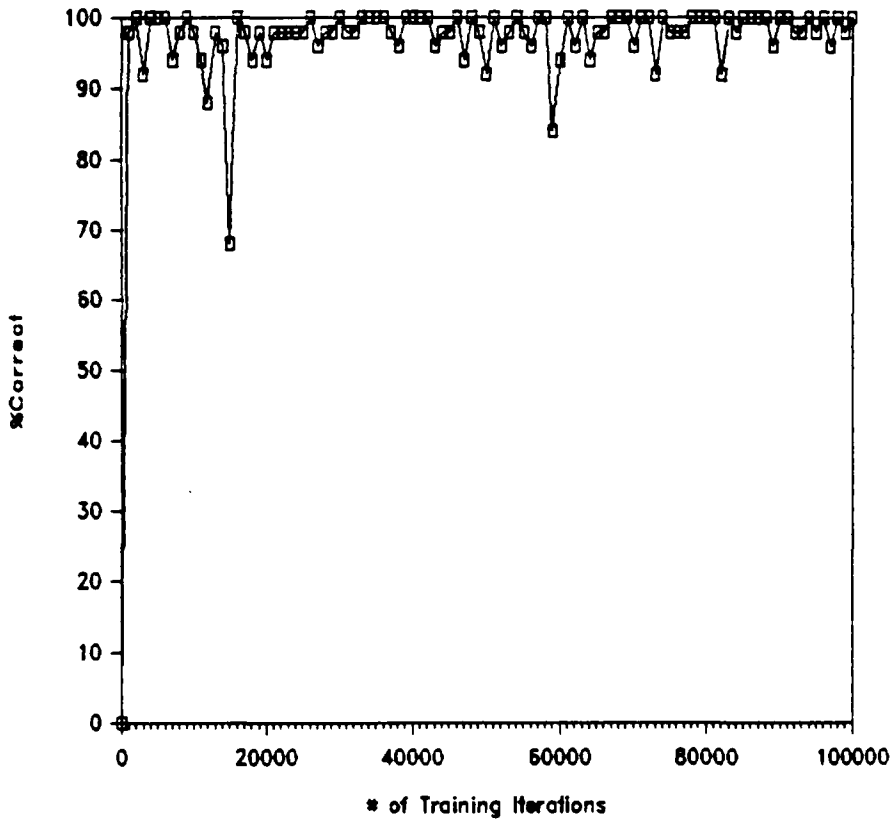


Figure 59. Training Accuracy for the Three Class Problem with the Order of the Class Values Switched.

55) that has been allowed to train the rate term within the sigmoid function. As seen in both Figures 60 and 61, that the sigmoid requires more training time than required for a hard limiter. This is because the sigmoid produces errors as it slopes from one data region to the next.

To illustrate that the neurolog can be used in a multi-layer perceptron configuration, a neural net shown in Figure 62 is constructed. The data consist of three classes with uniformly distributed data as shown in Figure 63. All the neurologs in the net used a constant rate (β) of 5.0. Figure

64 shows the accuracy versus training time.

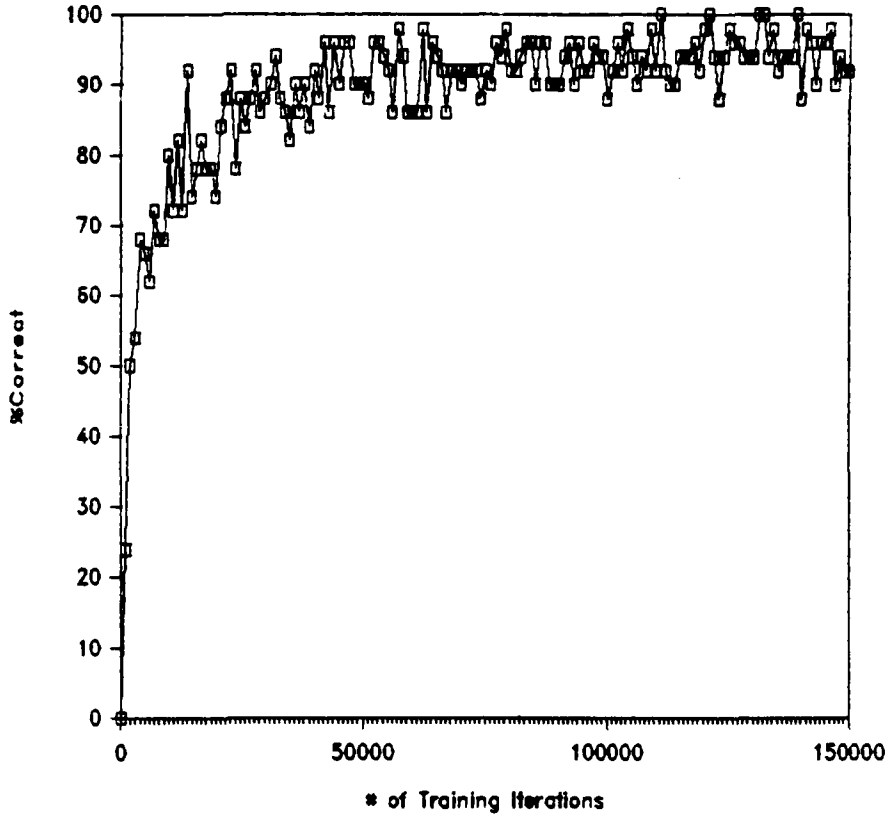


Figure 60. Training Accuracy Using the Sigmoid Function for the Exclusive-Or Problem.

Conclusions and Recommendations

The improvements made for the basic neuron has increased the computational power of a neuron. The ability to connect the neurologs in a multi-layer perceptron configuration will allow a decrease in the number of functions that any one neurolog needs.

Overall, the neurolog should increase the computational power of neural networks. Its capability to create complex hypersurfaces will obviously assist in data classification

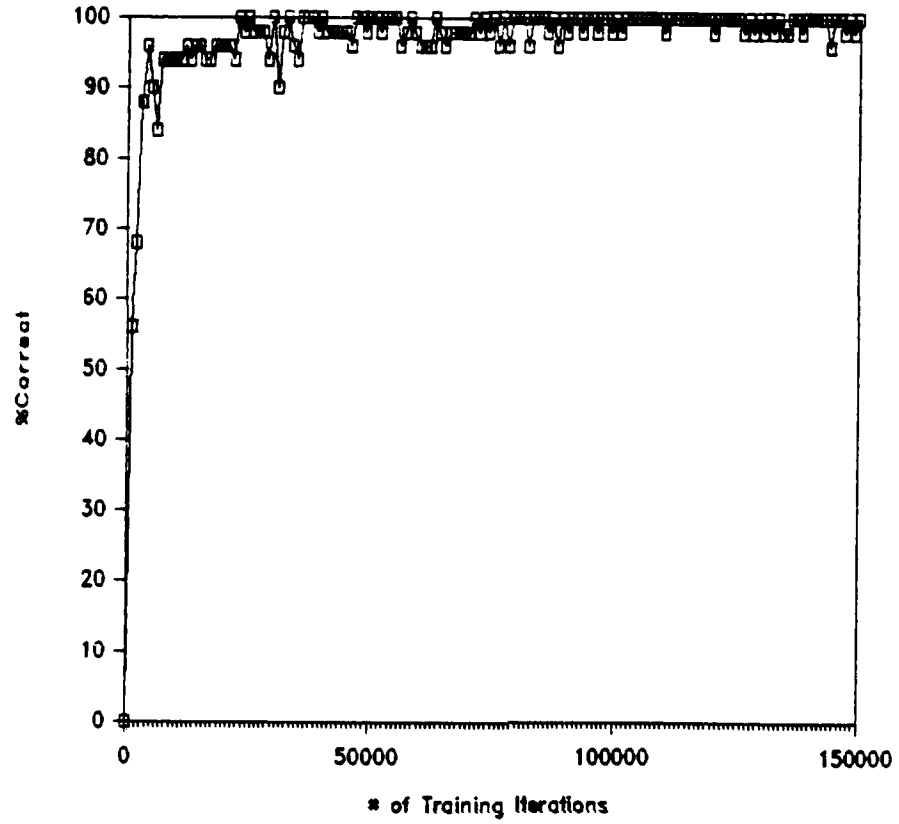


Figure 61. Training Accuracy Using Rate (β) Training for the Exclusive-Or Problem.

and help decrease net size. Multi-level outputs along with the capability to include random number generators as possible functions, should add new dimensions (such as guessing, the study of chaos, game playing, and so on) as neural networks are required to solve increasingly complex problems.

However, more investigations will be required in this area to determine what functions may be required to solve even a modest problem. Perhaps a search algorithm could be developed to assist in finding the functions that may be required by the neurolog. Also, research will be needed to

determine how many classes could be solved by one neurolog since it is possible to add more sigmoids or hard limiters together to provide additional multi-level outputs. Finally, feedback or lateral inhibition may be possible since these "new" inputs are no more than a complex function of the "old" inputs.

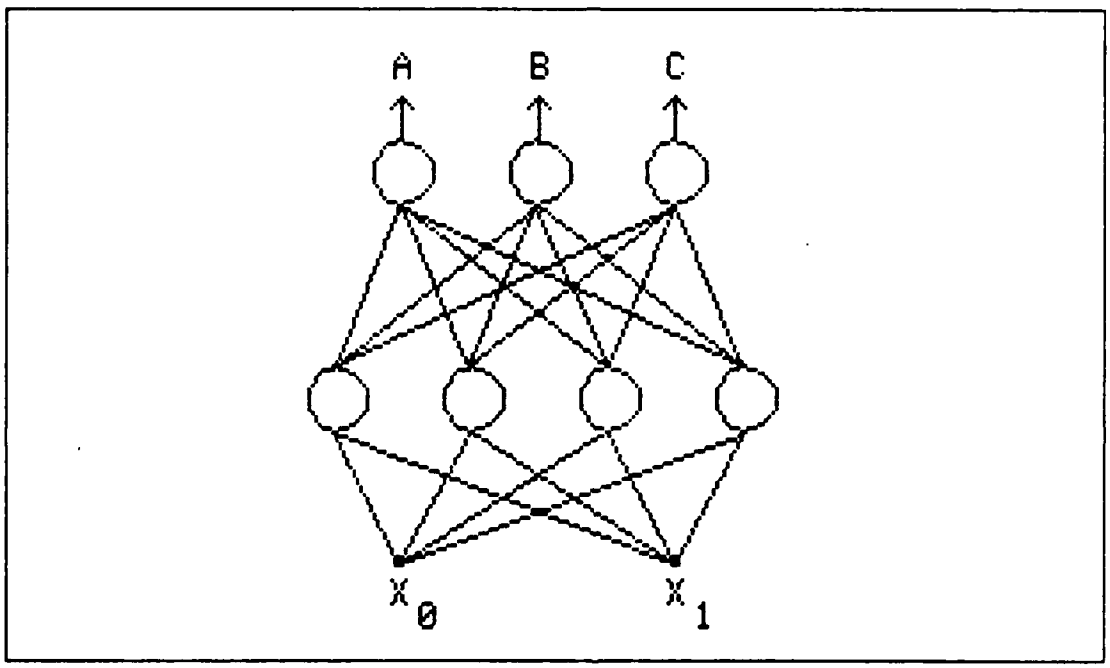


Figure 62. Neurologs in a Multi-Layer Perceptron Configuration.

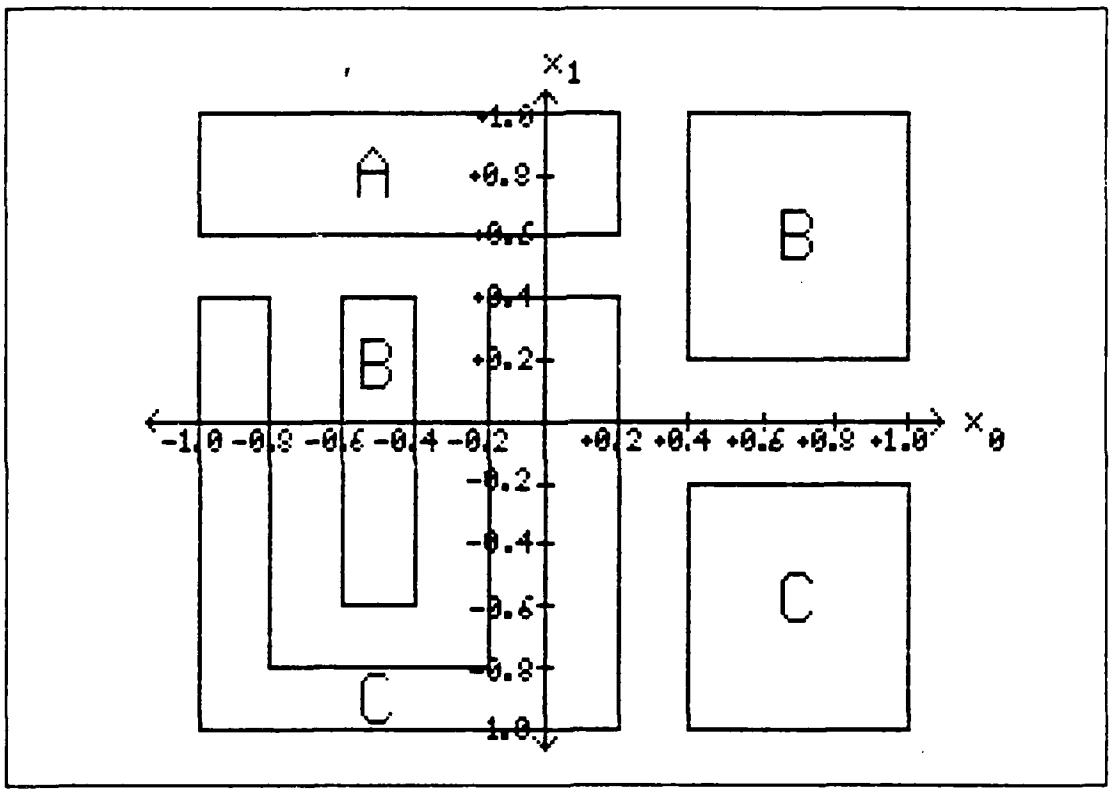


Figure 63. Data for the Multi-Layer Perceptron Configuration Test.

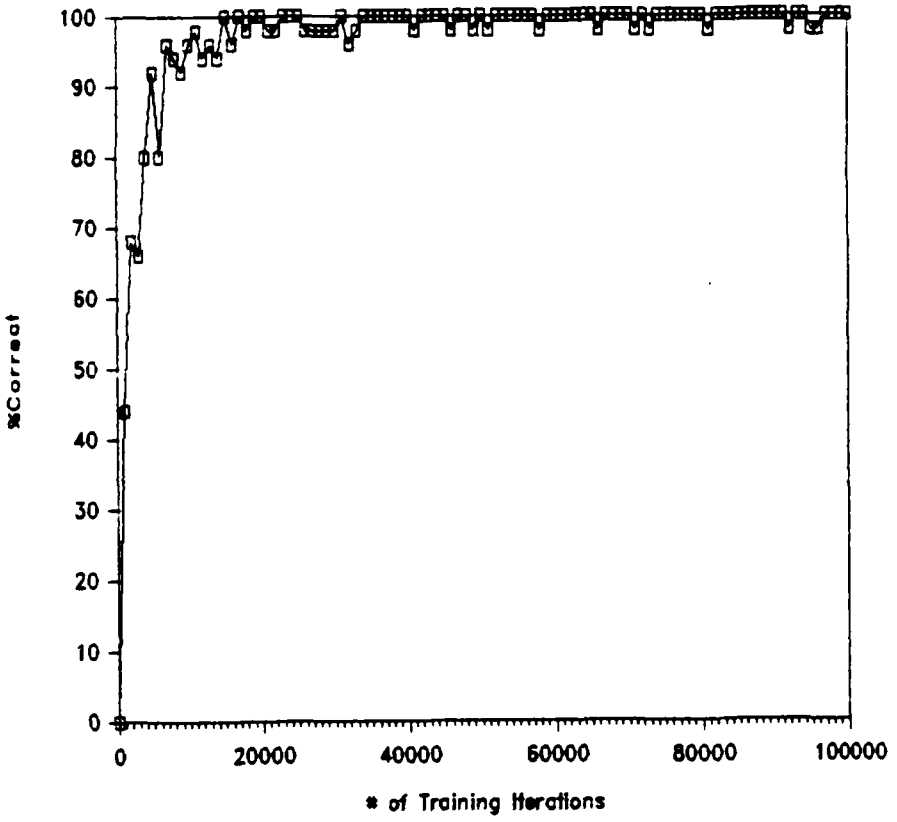


Figure 64. Training Accuracy for a Multi-Layer Perceptron Configuration.

VIII. Conclusion

Because of the numerous problems researched in this thesis, recommendations and conclusions for each topic can be found at the end of the corresponding chapter.

Bibliography

1. Cox, Kevin S. An Analysis of Noise Reduction Using Backpropagation Neural Networks. MS Thesis, AFIT/GE/ENG/88D-3. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

2. Dahl, Edward D. "Accelerated Learning Using The Generalized Delta Rule," IEEE First International Conference On Neural Networks, 2: 523-530 (June 1987).

3. Greenwood, Dan. "Netrologic Neural Networks Tutorial." Tutorial prepared by Netrologic, San Diego, CA.

4. Kohonen, Teuvo. "Self-Organizing Maps," Tutorial from the IEEE First Annual International Conference On Neural Networks. San Diego, CA. 21 - 24 June 1987.

5. Lippmann, Richard P. "An Introduction to Computing with Neural Networks," IEEE Acoustics Speech and Signal Processing, 4: 4-22 (April 1987).

6. Potter, Ralph K. and others. Visible Speech. New York: D. Van Nostrand Company, Inc., 1947.

7. Rabiner, Lawrence R. and Ronald W. Schafer. Digital Processing Of Speech Signals. Englewood Cliffs NJ: Prentice-Hall, Inc., 1978.

8. Rogers, Steven K. and James Stright. "How Backward Error Propagation Reduces Error." Class Handout in EENG 621, Pattern Recognition II. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 27 April 1988.

9. Rogers, Steven K., Instructor. Personal Interviews. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, April 1988.

10. Rosati, John J. and others. "Artificial Neural Systems (ANS) And Neural Computing." Tutorial prepared by the Technology Training Corporation, April 1988.

11. Ruck, Capt Dennis W. Multisensor Target Detection And Classification. MS Thesis, AFIT/GE/ENG/87D-56. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A177598).

12. Shepanski, J. F. "Fast Learning in Artificial Neural Systems: Multilayer Perceptron Training Using Optimal Estimation," IEEE International Conference On Neural Networks, 1: 465-472 (July 1988).

13. Tamura, Shin'ichi and Alex Waibel. "Noise Reduction Using Connectionist Models," IEEE International Conference On Neural Networks, 1: 553-556 (April 1988).

14. Yang, Hedong and Clark C. Guest. "Performance of Backpropagation for Rotation Invariant Pattern Recognition," IEEE First International Conference On Neural Networks, 4: 365-370 (June 1987).

Vita

Captain Mark K. Lutey [redacted] He graduated from Cuyahoga Falls High School, Cuyahoga Falls, Ohio in 1974. Captain Lutey received a Bachelor of Science degree in Electrical Engineering from South Dakota State University in May 1983. Upon graduation he received a commission in the USAF and was assigned to Space Command, Buckley ANGB, Colorado. Captain Lutey entered the Masters Program in the School of Engineering, Air Force Institute of Technology, in June 1987.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-23	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

PROBLEM SPECIFIC APPLICATIONS FOR NEURAL NETWORKS (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Mark K. Lutey, Captain USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1988 December	111

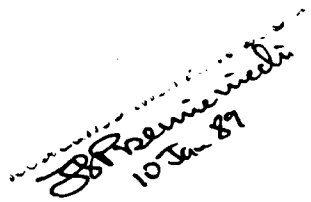
16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Speech Recognition, Multi-Layer Perceptrons, Neural Network, Kohonen Network, Neurolog, Accelerated Learning For Multi-Layer Perceptrons
12	09		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: Matthew Kabriski, PhD

Professor of Electrical Engineering



20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Dr. Matthew Kabriski Professor, GS-15	(513) 255-5276	AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

Continued from Block 19: Abstract

The purpose of this thesis is to examine several topics relating to neural networks. First, the investigation of the error output for a multi-layer perceptron is examined to determine if the error calculation can be modified to decrease the training time. The result is a slight improvement.

Next, the sigmoid function usually used by multi-layer perceptrons is investigated to determine if modifying terms within the sigmoid function will decrease training time. An improvement is found in the performance and in some cases by as much as an order of magnitude.

Then the subject of adding an additional class to the problem space is examined. Regardless of the data class added or the network size, there is no advantage to using a previously trained multi-layer perceptron as a starting state to be trained additionally to include the new data class. Nothing is gained compared to starting the network form its untrained state.

This is followed by an investigation to determine whether a multi-layer perceptron can be used to reduce noise added to a signal. Results show that the multi-layer perceptron, when trained with three specific frequencies, reduced noise but would "resonate" at these frequencies only. The network may also be limited to the number of individual frequencies on which it will perform noise reduction.

Next, the combination of using a Kohonen self-organizing feature map and a multi-layer perceptron to perform isolated word recognition is tested. An 80 per cent accuracy is achieved for speaker dependent, isolated word recognition. Accuracy falls to approximately 40 per cent for speaker independent, isolated word recognition.

Finally, an improvement in the basic neuron element usually used by the single and multi-layer perceptrons is presented. Results show that the computational power for a single neuron is greatly enhanced and that use in a multi-layer perceptron configuration is still possible.